Electronic Colloquium on Computational Complexity, Report No. 10 (2005)

Almost Completeness in Small Complexity Classes

Olivier Powell *

Abstract

We constructively prove the existence of *almost complete* problems under logspace many-one reduction for some *small complexity classes* by exhibiting a parametrizable construction which yields, when appropriately setting the parameters, an almost complete problem for PSPACE, the class of space efficiently decidable problems, and for SUBEXP, the class of problems decidable in subexponential time. Our construction also implies the existence of almost complete problems under logspace manyone reductions for bigger classes, such as E or EXP. We also investigate almost completeness for smaller time complexity classes, such as P and QP, and clearly identify and explain the reasons (which are not the same for P and QP) why our approach fails for these complexity classes.

Keywords: computational complexity, quantitative complexity, resource bounded measure, martingales, logspace manyone reductions.

1 Introduction

In [Lut92], Lutz introduced Resourced Bounded Measure ($\mathcal{RBM}$), which permits to give a size (big or small) to the *measurable* subsets of typical deterministic time bounded big complexity classes, such as E, the class of problems decidable in exponentially linear time. Lutz's $\mathcal{RBM}$ not only permits to describe the quantitative properties of some sets, it also enables the introduction of new and important concepts, the most studied of them being perhaps *weak completeness*, introduced in [Lut95]. Almost (and weakly) complete problems are a refinement of the well known definition of complete problems. Intuitively, a problem is complete for a given complexity class if it captures *all the hardness* in it, in the sense that any good solution (algorithm) to a complete problem yields a good solution for any problem of the class. In comparison, a good solution to an almost complete problem should yield a good solution not for *every* problem in the complexity class, but for *many* of them. The meaning of *many* has to be made precise, and it is done by using Lutz's Resource Bounded Measure ($\mathcal{RBM}$). Weakly complete problems are defined similarly as almost complete problems, but by replacing *many* with *a non negligible amount*.

The fairly abundant literature treating almost and weak completeness ([LM94], [Lut95], [Jue95], [JL95a], [JL95b], [LM96], [ASMZ96], [ASTZ97], [AS00], [ASMRT03], [Pow03]) is a testimony of the interest raised by these notions in the structural/quantitative complexity theory community. The main interests have been in understanding the different notions of almost and weak completeness with respect to different notions of reductions, such as length-increasing, manyone, bounded truth-table, truth-table, and Turing reductions, and full classification is close (c.f. the end of [ASMRT03] for a state of the art on that subject).

We follow the standard convention of calling a complexity class *big* if it contains E (problems decidable in exponentially linear time), and *small* if it is not known to contain E. Whereas knowledge has grown a lot on the subjects of weak and almost completeness, this only remains true when regarding *big* complexity classes: to our knowledge, there exist no other almost (or weak) completeness results than the one from [Pow03] when it comes to *small* complexity classes.

*Université de Genève, Centre Universitaire d'Informatique, rue du Général Dufour 24, CH-1211 Genève 4, Switzerland, olivier.powell@cui.unige.ch

ISSN 1433-8092

This state of affair holds despite the fact that typical representatives of small complexity classes are P and PSPACE which, as their nick name of time (or space respectively) efficient decidable problems suggests, are arguably those of highest interest. The reason for this is not the lack of intrinsic interest in studying almost and weak completeness in small complexity classes, but probably rather the fact that Lutz's $\mathcal{RBM}$ does not work well in small complexity classes (c.f. [Pow04] for a discussion on this issue).

This paper investigates the matter of almost completeness in small complexity classes, and partially fills the gap by proving the existence of almost complete problems under logspace manyone reductions for PSPACE and SUBEXP. The construction we make is parametrizable, so as a corollary, we also prove that there exist almost complete problems for E and EXP under *logspace* manyone reductions. This last result differs from a previous similar results from [ASMRT03], which shows the existence of almost complete problems for E and EXP under *polynomial time* manyone reductions (whereas we consider logspace manyone reductions). Our proof is inspired by the construction of [ASMRT03], and to overcome the fact that Lutz's $\mathcal{RBM}$ is not known to work for small complexity classes we use a modernized version of the construction of [AS94], in which a definition of an $\mathcal{RBM}$ *à la Lutz* is defined for P (and other small complexity classes), by taking to our advantage the insight given by the maturity that general $\mathcal{RBM}$ theory has gained in a decade.

We have taken special care in letting our results be as general as possible by keeping them parametrizable. For example, we prove in theorem 4.35 that any complexity class satisfying some minimal conditions admits an almost complete set under logspace manyone reductions. Then, setting appropriately the parameters gives almost complete sets for some well known complexity classes. As we emphasised, our main objective has been to obtain almost complete problems for small complexity classes: the smaller the class the better, with an ideal objective of reaching the level of P. This we could not do, but the fact that we kept our proofs as general as possible permits to clearly identify where the construction fails for P, thus hopefully helping further researches which would investigate the matter of almost completeness in P. While P stays out of reach at present, results concerning QP would perhaps temporarily spare the impatient a hopefully not too long wait. What we said here above about P is also true for QP: we clearly identify the reasons why our construction fails for this complexity class, but perhaps surprisingly, the reasons are not the same as those for P .

2 Conventions

We follow standard definitions of complexity theory, see for example [Pap94] or [BDG94a] and [BDG94b]. The set $\{M_i\}_{i\in\mathbb{N}}$ is an effective enumeration of Turing machines. $\{0,1\}^\infty$ is the set of (infinite binary) sequences. $\{0,1\}^*$ is the set of finite (binary) words. For $x \in \{0,1\}^* \cup \{0,1\}^\infty$, $|x|$ is the the length of x. For $N \in \mathbb{N}$, $\{0,1\}^N$ is the set of words of length N and $\{0,1\}^{\leq N}$ the set of words of length at most N. λ is the empty word, of length 0. When it is obvious what is the word x considered, we sometimes implicitly use the capital letter N instead of $|x|$. We put the canonical (length-lex increasing) ordering on words and the symbol s_i is used for the ith word under that ordering. Therefore, we have: $s_1 < s_2 < s_3 < s_4 < \ldots$, with $s_1 = \lambda$, $s_2 = 0$, $s_3 = 1$, $s_4 = 00$, $s_5 = 01$, etc... If x is a word or a sequence and for $1 \leq i \leq j \leq |x|$, $x[i,j]$ is the word consisting of the ith through jth bits of x. We use the two following shortcut notations: $x[i] := x[1,i]$ and $x(i) := x[i,i]$. If x is a word and y is either a word or sequence such that $x = y[1,|x|]$, we say that x is a prefix of y, which we write $x \sqsubseteq y$. For $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ with $m \leq n$, we say that y is a suffix of x, which we note $x \sqsupseteq y$, if $x[n-m+1,n] = y$ If $\mathcal{C}$ is a set, we write $2^{\mathcal{C}}$ for the power set of $\mathcal{C}$, and thus $2^{\{0,1\}^*}$ is the power set of $\{0,1\}^*$. A language is a set of words, and thus a point of $2^{\{0,1\}^*}$. *Problem* and *language* are synonyms. The characteristic sequence χ_L of a language L is the sequence defined by $\chi_L(i) = 1$ if $s_i \in L$ and $\chi_L(i) = 0$ if $s_i \notin L$. The mapping from $2^{\{0,1\}^*}$ to $\{0,1\}^\infty$ which maps a language L to its characteristic sequence is one-to-one and onto. Because of this bijection, we take the liberty to see languages as sets of words, points of $2^{\{0,1\}^*}$ or points of $\{0,1\}^\infty$, and we freely skip from one point of view to another. We use the standard definition of manyone reductions from a language to another. Unless explicitly stated otherwise,

a reduction is always a logspace manyone reduction. We use the notation $A \leq B$ to say that the language A reduces to B. The (lower) span of a language A is $P_m^{\leq}(A) := \{B \mid B \leq A\}$. We use standard complexity classes, except for SUBEXP, the class of problems decidable in sub-exponential time which is probably less common.

Definition 2.1 *Fix $i, j \in \mathbb{N}$, the jth slice of* SUBEXP *is* $\mathsf{E}_{\mathsf{j}} = \bigcup_{i \in \mathbb{N}} \mathtt{DTIME}(2^{\mathcal{O}(N^{\frac{i-1}{ij}})})$*, and* $\mathtt{SUBEXP} = \bigcap_{j \in \mathbb{N}} \mathsf{E}_{\mathsf{j}}$.

3 Resource Bounded Measure

A Central tool of Lutz's $\mathcal{RBM}$ is martingales, which are mappings from words to integers that have the property, informally stated, that they describe the cash amount of a gambler while playing the so called *casino game*. We give in the beginning of section 3.2 a short and informal description of the casino game. For more detailed explanations, see [Lut97] or [ASMRT03]. Informally, Lutz's definition requires that, in order to obtain an $\mathcal{RBM}$ in a complexity class $\mathcal{C}$ of *somehow efficiently* decidable problems, one considers *logarithmically faster* computable martingales. This logarithmic requirement implies that, for small complexity classes, martingales should become computable in sublinear time, and this brings technical difficulties. For more explanations, and different propositions to overcome them, see [May94], [AS94], [AS95], [Str97], [Mos02] or [Pow04]. To obtain our results, we need to have $\mathcal{RBM}$'s for small complexity classes. We shall use a revised and modernized version of the $\mathcal{RBM}$ for P proposed in [AS94], which we describe in subsection 3.3. Our job has been simplified by [Mos02] which has previously adapted the work from [AS94] to obtain notions of $\mathcal{RBM}$ for probabbilistic classes. To do so, we need to describe, in subsection 3.2, the type of martingales which we will need to consider, which we call *good* (or Γ) martingales. As suggested, these martingales will have, amongst other things, to be computable in sublinear time and we therefore specify in the next subsection some details about the computational model we shall use.

3.1 Computational Model

We need to consider Turing machines ($\mathcal{TM}$) computing in sublinear time. With the usual model, such a $\mathcal{TM}$ never has access to the rightmost bits of its input band: moving the read head to this point already takes linear time. This motivates the use of a random access memory (RAM) Turing machine.

Definition 3.1 *A $\mathcal{TM}$ M has* RAM *access to its input if it has a special* query tape *and a special* query sate. *When M writes s_i, the ith word of $\{0,1\}^*$ on its query tape and enters the query state, it gains knowledge of the content of the ith position of the input tape in unit time. When this happens, we say that s_i has been queried by M.*

Convention 3.2 *When about the space complexity of a* RAM *$\mathcal{TM}$, we do not take into account the input tape(s), the output tape, nor the query tape.*

From now on, any $\mathcal{TM}$ is considered to be a RAM $\mathcal{TM}$. Given a $\mathcal{TM}$ M and an input x, we can consider the set of the words M queries during its computation on input x. We define a notion of *good* $\mathcal{TM}$'s (c.f. definition 3.5 below), based on the property of not querying too many different words, even when we let the input range over all words of a given size. This is done by asking the *good* $\mathcal{TM}$'s to query words only inside a (small) set called a *dependency set*.

Definition 3.3 *A dependency set $Q = \{Q_i\}_{i \in \mathbb{N}}$ is a chain $Q_0 \subseteq Q_1 \subseteq Q_2 \subseteq \ldots$ of finite languages with $Q_N \subseteq \{s_i\}_{1 \leq i \leq N}$ for all $N \in \mathbb{N}$.*

Definition 3.4 *Let $Q = \{Q_N\}_{N\in\mathbb{N}}$ be a dependency set, and M be a $\mathcal{TM}$. M is said to* query its input in Q *if for every $N \in \mathbb{N}$ it holds that:*

$$\bigcup_{x\in\{0,1\}^N} \{\omega \in \{0,1\}^* \mid T(x) \text{ queries } \omega \text{ at some point during its computation}\} \subseteq Q_N$$

We can now define what a *good* or Γ $\mathcal{TM}$ is: a $\mathcal{TM}$ which computes quickly, and queries little, even when the input ranges over every possible input of bounded size.

Definition 3.5 *Let f be a time (or space [1]) bound function. A $\mathcal{TM}$ M is a $\Gamma(f)$ $\mathcal{TM}$ if M computes in time (or space respectively) $f(N)$, and M queries its input in a dependency set $Q = \{Q_N\}_{N\in\mathbb{N}}$ with $|Q_N| \leq f(N)$.*

By extension, if $\mathcal{F}$ is a family of time or space bounds, we say that a $\mathcal{TM}$ M is a $\Gamma(\mathcal{F})$ $\mathcal{TM}$ if $\exists f \in \mathcal{F}$ such that M is a $\Gamma(f)$ $\mathcal{TM}$. The purpose of introducing Γ computations may not be clear yet, but we shall comment on this again in subsection 3.3. The idea, which comes from [AS94], is to restrict the computational power allowed to compute martingales by asking for Γ computations. This will enable to construct the $\mathcal{RBM}$ for small complexity classes. More on this later.

Sometimes it will be crucial for us that a $\mathcal{TM}$ is able to compute the length of its input. More precisely, it will be crucial for us that a $\mathcal{TM}$ M, on input x, is capable of computing $s_{|x|}$ (which is almost equivalent to computing $|x|$, but we leave these details to the reader). It is fairly easy to see that a $\mathcal{TM}$ can compute very efficiently (in time $\mathcal{O}(\log N)$) the length of its input (c.f. [Bus87]). However, this is not so for $\Gamma(\mathcal{O}(\log N))$ $\mathcal{TM}$'s. Indeed: if we consider Q_N to be the words queried by a $\mathcal{TM}$ M computing the length of its input, (for any input in $\{0,1\}^N$), we do not have the property that $Q_N \subseteq Q_{N+1}$, i.e. the family $\{Q_i\}_{i\in\mathbb{N}}$ is not a dependency set. So to consider the smallest dependency set in which M queries, we have to consider $Q'_0 = Q_0$ and $Q'_{N+1} = Q_{N+1} \cup Q'_N$. $\{Q'_N\}_{N\in\mathbb{N}}$ is a dependency set, but it is not small. Therefore M, although computing in $\mathcal{O}(\log N)$ time is *not* a $\Gamma(f(N))$ $\mathcal{TM}$ for a reasonable f. For this reason, we will sometimes consider $\mathcal{TM}$'s with an auxiliary input tape, having the property that on input x the auxiliary input tape is automatically initialised with the binary encoding of $s_{|x|}$.

Convention 3.6 *When we want to explicitly show that a $\mathcal{TM}$ M has an auxiliary input tape which, on input x, automatically gets initialised to $s_{|x|}$, we replace the symbol M with $M^{|x|}$. We say that a function d is a $\Gamma^{|x|}(f)$ function if there exists a $\Gamma(f)$ $\mathcal{TM}$ $M^{|x|}$ computing d.*

For completeness, let us specify how $\mathcal{TM}$'s output rational numbers.

Definition 3.7 *A $\mathcal{TM}$ $M : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ is said to compute a rational function $f : \{0,1\}^* \to \mathbb{Q}$ if $\forall x \in \{0,1\}^*$, the output (a, b) of M, interpreted as (the binary encoding of) a pair of integers satisfies $f(x) = \frac{a}{b}$.*

It was shown in [Pow03] that this convention permits to efficiently compute easy arithmetics on $\mathbb{Q}$.

Lemma 3.8 ([Pow03]) *$\exists c$ a constant such that for any family of rational functions $\{f_i\}_{1\leq i\leq m}$ such that $\forall i$ f_i is computable in $g(N)$ space, simple arithmetics on the f_i's (sums, subtractions, multiplications and divisions) can be carried out in $c[g(N) + \log(m)]$ space.*

3.2 Martingales

To make the story short, a classical result from [Vil39] shows that Lebesgue measure can be described in terms of martingales. From this observation, Lutz developed *Lutz's Resource Bounded Measure* ($\mathcal{RBM}$). The idea is the following: if instead of allowing martingales to be any "purely

[1] When we want to explicitly show whether we are talking about a space or time bound Γ-function, we use the notation Γ_s or Γ_t respectively.

mathematical" function, we require them to be computable, we obtain a zoomed version of the Lebesgue measure. The higher we want the zoom factor to be, the more efficiently we have to require the martingales to be computable. This zooming process is limited, and we cannot zoom above a certain factor[2]. Informally, the reason is that when the computational power given to martingales becomes sublinear, martingales do not have the time to read the whole of their input, which gives rise to technical difficulties notably preventing us in the present state of the art, from proving that the 3rd *measure axiom*[3] holds. Intuitively, the problem when defining an $\mathcal{RBM}$ in a small complexity class $\mathcal{C}$, is that martingales still look too powerful: we cannot prove that $\mathcal{C}$ itself is not small. To bypass this limit, different variations of Lutz's $\mathcal{RBM}$ have been developed ([May94], [AS94], [AS95] and [Str97]). We give in this section the definitions which will permit to describe an $\mathcal{RBM}$ powerful enough (in terms of zooming factor) for measuring in small complexity classes such as `P` or `PSPACE`. This measure is the one defined in [AS94], but we revisit it and modernize it, taking advantage of the improvements general $\mathcal{RBM}$ has enjoyed since [AS94] was published. As we may have hinted previously, the main idea of [AS94] is to further restrict the power of martingales by asking them to be not only efficiently computable, but by machines not querying their input too much, i.e. by Γ machines.

We start by reminding the reader of the definition of martingales. Martingales can be thought of intuitively as describing the course of the so called *casino game*. Informally, the casino secretly chooses a language L which it does not reveal to the gambler, and the gambler has an initial cash amount of one dollar. The casino and the gambler play the following infinite fair game: at each round, (say the Nth round for example), the player is allowed to wager a stake σ which may not exceed his total current capital (the casino makes no credit...), gambling on whether the Nth word s_N belongs to the language L or not. The casino then reveals the value of $\chi_L[N]$, and if the prediction of the gambler was right the casino pays the gambler σ dollars (the casino is supposed to have access to an infinite capital, so it never goes bankrupted), and the gambler has thus doubled its stake σ. Otherwise, the gambler loses his stake. It may be pointed out that this game is not "memoryless": at the Nth stage of the game, the gambler recalls the past history of the game, i.e. he has access to $\chi_L[1, N-1]$. A martingale is a function which describes the total cash amount of the gambler while playing the casino game. The informal description here above mathematically translates to the following definition.

Definition 3.9 *A super-martingale is a function $d : \{0,1\}^* \to \mathbb{R}^+$ such that $d(\omega) \geq \frac{d(\omega 0)+d(\omega 1)}{2}$, and a strict martingale is a special case of a super-martingale where the inequality above holds with equality.*

Intuitively, a super-martingale corresponds to a modification of the casino game where the player is allowed to throw away part of his capital at each round of the game. This may seem to be nonsense, but it becomes interesting when the martingale needs to be efficiently computed: informally, keeping count of "every single penny" may be computationally too expensive, so it may be more effective to throw away a bit of money... Usually the word martingale is used for strict martingales. Since in this paper we shall be dealing mainly with super-martingales, we separate ourselves from standard terminology by taking the following convention.

Convention 3.10 *A martingale is a super-martingale, and not a strict martingale.*

We say that a martingale d *covers* a language L if it is unbounded on the prefixes of (the characteristic sequence of) L, or equivalently, we say that L belongs to the success set of d. This notion is important: intuitively, a set is of null measure if it is a subset of the success set of a martingale (more on this in section 3.3).

Definition 3.11 *The success set $S^\infty[d]$ of a martingale d is defined by*

$$S^\infty[d] = \{L \subseteq \{0,1\}^* \mid \limsup_{N\to\infty} d(L[N]) = \infty\}$$

[2]More precisely, it is unknown if Lutz's $\mathcal{RBM}$ works for small complexity classes, i.e. classes which do not (or are not known to) contain `E`.

[3]c.f. section 3.3

Perhaps it is time to explain why we use super-martingales only, and not strict martingales. We do not want to say too much about this problem yet, since it will be discussed thoroughly in the next subsection, but in order to ensure that the measure we shall define has the property that "reasonable" unions of null sets are null sets too, we shall need the version of the *exact computation lemma* given below. Now if we were to have considered "strict" martingales instead of super-martingales, we could not have this lemma to hold: if, in the statement of the lemma, d was a strict martingale, $\hat{d}$ would still be a super-martingale, thus we could not ensure that any "approximate"martingale can be replaced by an "efficiently computable" martingale with a success set at least as large. The curious reader may wonder why the somehow usual trick used to transform a super-martingale into a strict martingale, which consists of replacing $\hat{d}$ by a function $\check{d}$, defined by $\check{d}(\omega[1,N]) = \check{d}(\omega[1,N-1]) + \frac{1}{2}(\hat{d}(\omega[1,N-1]w[N] - \hat{d}(\omega[1,N-1](1-w[N]))))$ does not work in this case. The reason is that the $\mathcal{O}(N)$ terms summation which is not a problem for $\mathcal{RBM}$ in big complexity classes is not acceptable for small complexity classes. Next we give the exact computation lemma which says, intuitively, that when a martingale can be approximated, we can replace it by a martingale which is exactly computable.

Lemma 3.12 (Exact computation) *Let d be a martingale such that there exists $\tilde{d}$ a $\Gamma^{|x|}(f)$ function approximating d in the following sense: $|d(\omega) - \tilde{d}(\omega)| \leq \frac{1}{|\omega|^2}$, then there exists $\hat{d}$ a $\Gamma^{|x|}(\mathcal{O}(f))$ martingale such that $S^\infty[d] \subseteq S^\infty[\hat{d}]$.*

Proof. Let $\bar{d}(\omega) := \tilde{d}(\omega) + \frac{4}{|\omega|}$. $\bar{d}$ is a super-martingale and $S^\infty[d] \subseteq S^\infty[\bar{d}]$, indeed: it is immediate that if $\bar{d}$ is a super-martingale, then its success set contains the success set of d, since $\bar{d}(\omega) = \tilde{d}(\omega) + \frac{4}{|\omega|} \geq d(\omega) - \frac{1}{|\omega|^2} + \frac{4}{|\omega|} > d(\omega)$. To see that $\bar{d}$ is a super-martingale, consider the following inequations, where $N = |\omega|$: $\frac{\bar{d}(\omega 0)+\bar{d}(\omega 1)}{2} = \frac{\tilde{d}(\omega 0)+\tilde{d}(\omega 1)}{2} + \frac{4}{N+1} \leq \frac{d(\omega 0)+d(\omega 1)}{2} + \frac{1}{(N+1)^2} + \frac{4}{N+1} \leq d(\omega) + \frac{1}{(N+1)^2} + \frac{4}{N+1} \leq \tilde{d}(\omega) + \frac{1}{N^2} + \frac{1}{(N+1)^2} + \frac{4}{N+1} = \bar{d}(\omega) - \frac{4}{N} + \frac{1}{N^2} + \frac{1}{(N+1)^2} + \frac{4}{N+1} \leq \bar{d}(\omega) - \frac{4}{N(N+1)} + \frac{2}{N^2} \leq \bar{d}(\omega)$. To finish the proof, we only need to notice that $\bar{d} \in \Gamma(\mathcal{O}(f))$ follows from the fact that $\tilde{d} \in \Gamma(f)$. □

3.3 Resource Bounded Measure

A $\mathcal{RBM}$ μ on a complexity class $\mathcal{C}$ is a partial function from $2^{\mathcal{C}}$, the power set of $\mathcal{C}$, to $\{0,1\}$. I t defines a notion of small and large sets in $\mathcal{C}$: a subset of $\mathcal{C}$ is called *small* or *null* if it is mapped to 0, *large* if it is mapped to 1, and *non-measurable* otherwise. In order for an $\mathcal{RBM}$ to be interesting, it should probably also be that some intuitively small sets are indeed of null measure, but most of all, in order to justify the fact of calling μ a measure, it should satisfy the following *measure axioms*:

M1 Easy unions of null sets are null sets.[4]

M2 Singleton sets are null sets.

M3 The whole space $\mathcal{C}$ is not a null set.

In [AS94], an $\mathcal{RBM}$ for `P` is defined. This construction is parametrizable, and it also yields $\mathcal{RBM}$'s for other classes such as `PSPACE` and `SUBEXP`. Also, it yields a measure for `E` and other big complexity classes, which coincides with Lutz's $\mathcal{RBM}$. This justifies calling the $\mathcal{RBM}$ of [AS94] a *generalisation* of Lutz's $\mathcal{RBM}$ to small complexity classes, although their construction contains some arbitrary choices, (the definition of Γ computation), and thus cannot claim to be *the* generalization of Lutz's $\mathcal{RBM}$. Indeed , subsequent definitions of $\mathcal{RBM}$'s for small complexity classes attempts can be found in the papers [AS95] and [Str97]. Also, [May94] defines an $\mathcal{RBM}$ for `PSPACE`, which is incomparable to the ones defined in the series of papers [AS94], [AS95] and

[4]The meaning of *easy unions* is informal, but it should definitely include finite unions. In our case, it will be made precise in lemma 3.17 and corollary 3.21.

[Str97]. Finally, [Pow04] discusses the existence of what could be seen as a "perfect" generalization of Lutz's $\mathcal{RBM}$, by introducing the concept of a *random based* $\mathcal{RBM}$. For our purposes, the constructions of [AS94] will be sufficient. Rewritten in our notations it can be thus summarized:

Lemma 3.13 (A measure for PSPACE) *Let $\mathcal{F} = \bigcup_{i\in\mathbb{N}} \log^i(N)+i$ be the unions of polylogarithmic functions. The partial function $\mu_{\mathtt{PSPACE}} : 2^{\mathtt{PSPACE}} \dashrightarrow \{0,1\}$ defined hereunder is a well defined partial function satisfying the measure axioms, it thus defines an $\mathcal{RBM}$ measure on PSPACE.*

$$\mu_{\mathtt{PSPACE}}(\mathcal{A}) = \begin{cases} 0 & \text{if } \exists d \in \Gamma_s^{|x|}(\mathcal{F}) \text{ a super-martingale such that } \mathcal{A} \subseteq S^\infty[d]; \\ 1 & \text{if } \mu(\mathtt{PSPACE} \setminus \mathcal{A}) = 0. \end{cases}$$

Lemma 3.14 (A measure for SUBEXP) *Fix $j \in \mathbb{N}$ and let $\mathcal{F} = \bigcup_{i\in\mathbb{N}} 2^{\mathcal{O}(\log(N)^{\epsilon_{i,j}})}$ (where $\epsilon_{i,j} = \frac{i-1}{ij}$). The partial function $\mu_{\mathsf{E_j}} : 2^{\mathsf{E_j}} \dashrightarrow \{0,1\}$ defined hereunder is a well defined partial function satisfying the measure axioms, it thus defines a $\mathcal{RBM}$ measure on $\mathsf{E_j}$.*

$$\mu_{\mathsf{E_j}}(\mathcal{A}) = \begin{cases} 0 & \text{if } \exists d \in \Gamma_t^{|x|}(\mathcal{F}) \text{ a super-martingale such that } \mathcal{A} \subseteq S^\infty[d]; \\ 1 & \text{if } \mu(\mathsf{E_j} \setminus \mathcal{A}) = 0. \end{cases}$$

In order to prove that these lemmas hold, we need to show that the three *measure axioms* hold. We prove in lemma 3.17 that M1 holds. As explained, the $\mathcal{RBM}$'s here above are not new: they are a modernized version of the $\mathcal{RBM}$ from [AS94]. This is why we do not prove M2 and M3, which are easy, although we discuss these two points briefly and informally hereunder. M2 is merely a consequence of the fact that the martingales are powerful enough to beat any singleton set. As an example, to beat a language $L \in \mathtt{PSPACE}$, a martingale could play a double or quit strategy on words of $\{0\}^*$ only, thus keeping the queried set small. It is here that it is important that algorithms computing martingales should have access, on an input of length N, to s_N on an auxiliary tape (convention 3.6). We leave the details to the reader. Roughly speaking, M3 is proved by taking advantage of the fact that martingales query their input in a small dependency set. This enables, for any fixed martingale d, to construct a language L which is not in the success set of d, and which is easy to decide (i.e which is in PSPACE or $\mathsf{E_j}$ receptively). For example, suppose that we have $d \in \Gamma_t^{|x|}(\mathcal{F})$. We want to show that there is a language in $L \in \mathtt{PSPACE}$ which is not in $S^\infty[d]$. What we do is that we construct L a language that diagonalises against d, i.e. a language which defeats d every time d wagers on a word. We leave once again the details to the reader, but the fact that $d \in \Gamma_t^{|x|}(\mathcal{F})$ guarantees that L is in PSPACE too, which is proved similarly as in the case of "classical" Lutz's $\mathcal{RBM}$ theory, adding that we can bound the recursive calls to the algorithm computing d using the fact that this algorithm queries its input in a dependency set. It remains to define (definition 3.16) what an *easy union* is, and to show (lemma 3.17) that an easy union of null sets is a null set too. First, let us take the following simplifying convention.

Convention 3.15 *When it is clear from the context whether we are talking about $\mu_{\mathtt{PSPACE}}$ or $\mu_{\mathsf{E_j}}$ (for some $j \in \mathbb{N}$), we omit suffixes, and write μ instead of $\mu_{\mathtt{PSPACE}}$ or $\mu_{\mathsf{E_j}}$.*

Intuitively, a family of null sets is *easy* if there is a family of uniformly computable martingales covering it, and each of these computations should query in a single common small dependency set. In order to make our results general enough so that they can be applied to different complexity classes we need to let the parameters free, and only fix them when later when we want to consider some particular $\mathcal{RBM}$.

Definition 3.16 *A family $\{X_i\}_{i\in\mathbb{N}}$ is a $\Gamma(f)$ time or space family of null sets (respectively), if there is a family $\{d_i\}_{i\in\mathbb{N}}$ of martingales such that there exists $M^{|x|}$ a $\mathcal{TM}$ such that for any fixed $i \in \mathbb{N}$ and $\forall \omega \in \{0,1\}^*$ $M^{|x|}(i,\omega)$ computes $d_i(\omega)$ in $f(i,N)$ time or space (respectively), querying its input in a dependency set $G_i = \{G_{i,N}\}_{N\in\mathbb{N}}$ with $G_{i,N}$ of size $f(i,N)$.*

With this definition, we can state the fact that easy unions of null sets are null sets too as a corollary of the following parametrized lemma.

Lemma 3.17 *If $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a non-decreasing space or time bound function computable in space or time (respectively) $r(a,b) \leq f(a,b)$ and such that $f(a,b) \geq \log(a)\log(b)$, then for $\{X_i\}_{i\in\mathbb{N}}$ a $\Gamma(f)$ space or time (respectively) family of null sets there exists a single martingale $\hat{d} \in \Gamma(\text{poly}(f(\log N, N)))$ such that $\bigcup_i X_i \subset S^\infty[\hat{d}]$.*

Proof. Let $\{d_i\}_{i\in\mathbb{N}}$ be the family of martingales witnessing the fact that $\{X_i\}_{i\in\mathbb{N}}$ is a $\Gamma(f)$ family of null sets. Consider $d(\omega) = \sum^\infty c_i d_i(\omega)$ with $c_i = \frac{k}{2^{4i+f(i,2^i)}}$ for some $k \leq \frac{6}{\Pi^2}$. Notice that for all i, it holds that $d_i(\omega) \leq 2^{f(i,N)}$, where $N = |\omega|$. (This is so because any d_i may at most double its value when betting on a word, and it may only wager on words of the dependency set which is of size at most $f(i,N)$). It is easily verified that d is a martingale, and that $\bigcup_i S^\infty[d_i] \subseteq S^\infty[d]$. Now consider $\tilde{d}$ the following approximation to d: $\tilde{d}(\omega) := \sum_{i=0}^{\log N} c_i d_i(\omega)$.

Claim 3.18 *$\tilde{d}$ is a $\Gamma(\text{poly}(f(\log N, N)))$ function.*

To compute $\tilde{d}$, we need to compute the c_i's and d_i's, (for i from 0 to $\log N$), multiply them and sum them all. Since $\{d_i\}_{i\in\mathbb{N}}$ is a witness of $\{X_i\}_{i\in\mathbb{N}}$ being a $\Gamma(f)$ family of null sets, each of the elements of $\{d_i\}_{0\leq i\leq \log N}$ is (uniformly) computable in time or space (respectively) $f(i,N) \leq f(\log N, N)$ (since $i \leq \log N$ and since f is by hypothesis non-decreasing). To compute c_i, we need to compute $1/2^{4i}$ and $1/2^{f(i,2^i)}$ (with $i \leq \log N$): $1/2^{4i}$ is computable in $\mathcal{O}(\log N)$ time and $\mathcal{O}(\log\log N)$ space, $2^{f(i,2^i)}$ is computable in $\mathcal{O}((f+r)(\log N, N))$ time and $\mathcal{O}(\log((f+r)(\log N, N)))$ space, thus c_i is computable in $\mathcal{O}(\log(N) + (f+r)(\log N, N))$ time and $\mathcal{O}(\log(f(\log N, N)) + r(\log N, N))$ space. Thus the $c_i d_i(\omega)$'s (for $1 \leq i \leq \log N$) are (uniformly) computable in $\mathcal{O}((f+r)(\log N, N))$ time and space. Finally, to compute $\tilde{d}$, we need to sum the $c_i d_i$'s. Since these are rational numbers q represented as pairs of integers (a,b) with $q = \frac{a}{b}$, this will require simple arithmetics: additions and multiplications. Simple arithmetics (sums) of m rational functions computable in space or time $g(N)$ can be computed in $\text{poly}(mg(N))$ space or time. (In fact, this can be done even more efficiently in the case of space computations, c.f. lemma 3.8). In the case of computing $d(\omega) = \sum_{i=0}^{\log N} c_i d_i(\omega)$, we have $m = \log N$ and $g(N) = \mathcal{O}((f+r)(\log N, N))$. Thus $\tilde{d}$ is computable in $\text{poly}((f+r)(\log N, N)) = \text{poly}(f(\log N, N))$ time or space respectively, by a $\mathfrak{TM}$ querying its input in a dependency set of size $\log(N)f(\log(N,N)$ (since the $\{d_i\}_{i\in\mathbb{N}}$'s all query in a dependency set of size $f(\log N, N)$ by assumption), thus the claim is substantiated.

Claim 3.19 $|d(\omega) - \tilde{d}(\omega)| \leq \frac{1}{N^2}$

We easily verify this claim by developing the series for $d(\omega)$ and $\tilde{d}(\omega)$ and recalling that $\frac{d_i(\omega)}{2^{f(i,2^i)}} \leq 1$ for $i > \log N$, since as previously pointed out d may at most have wagered (and doubled its capital) $f(\log N, N)$ times at the Nth round of the casino game: $|d(\omega) - \tilde{d}(\omega)| = \sum_{\log(N)+1}^{\infty} c_i d_i(\omega) \leq \sum_{\log N}^{\infty} \frac{k}{N^4 2^{f(\log N, N)}} d_i(\omega) \leq \frac{k}{N^2} \sum_{\log N}^{\infty} \frac{1}{N^2} \leq \frac{k}{N^2}\frac{\Pi^2}{6} \leq \frac{1}{N^2}$.

The *exact computation lemma* 3.12 and the two claims 3.18 and 3.19 above ensure that there exists $\hat{d}$ a martingale as announced in the statement of the lemma. □

For the cases of the $\mathcal{RBM}$'s on PSPACE or (a slice of) SUBEXP from lemmas 3.13 and 3.14, we thus define "easy unions":

Definition 3.20 *A* PSPACE easy union *is a $\Gamma(f)$ family of null sets, with f a* polynomial *function of its first argument and a* polylog *function of its second argument. An $\mathsf{E_i}$* easy union *is a $\Gamma(f)$ family of null sets, with $f(k,N)$ a function in $2^{\mathcal{O}(k^{\frac{i-1}{ij}} + \log(N)^{\frac{i-1}{ij}})}$.*

This definition is such that it holds as a corollary to lemma 3.17 that finite unions of null sets are null sets too.

Corollary 3.21 *When considering* PSPACE *and μ_{PSPACE}, or alternatively $\mathsf{E_i}$ and $\mu_{\mathsf{E_i}}$, it holds that* easy unions *of nulls sets are null sets too, i.e. M1, the first measure axiom holds.*

Remark 3.22 *It follows from definition 3.16 that finite unions (of null sets) are easy unions.*

4 Abstract Construction of an Almost Complete Problem

In this section, we give ourselves all the tools that permit the construction of an almost complete problem for different complexity classes. We keep to the line of making statements (definitions, lemmas, theorems, etc...) general enough to let them be applied to different complexity classes by making results parametrizable. For the sake of comprehensiveness, let us sketch the way this section is organised. Subsection 4.1 is where we give the technical definitions we need so as to construct, in subsection 4.2, an almost complete language. The formal definition of an almost complete set is the following.

Definition 4.1 *Let $\mathcal{C}$ be a complexity class and μ be an $\mathcal{RBM}$ on $\mathcal{C}$. A problem A is almost complete for $\mathcal{C}$ if $A \in \mathcal{C}$, A is not complete for $\mathcal{C}$ (i.e. $\exists B \in \mathcal{C}$ such that $B \not\leq A$) but the span of A is large, i.e. $\mu(P_m^{\leq}(A)) = 1$.*

Sticking close to this definition, we construct, in section 4.2 and on parameter $\mathcal{C}$ a complexity class, two sets $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ such that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$, and we prove that, under reasonable conditions, these two sets belong to $\mathcal{C}$. Finally, in section 4.3, we state and prove the main theorem, which states some conditions under which $A_{\mathcal{C}}$ is an almost complete set foxr $\mathcal{C}$.

4.1 Setting the Context

As explained previously, we want to construct two sets B and A such that $B \not\leq A$, and the construction is made by diagonalization. So what do we diagonalize against? We diagonalize against an effective enumeration of reductions

Convention 4.2 *$\{R_i\}_{i\in\mathbb{N}}$ is an effective enumeration of reductions, and R_i is computable in $i\log(N) + i$ space.*

The existence of such an enumeration is standard. It can be proved by using a universal transducer together with a "yardstick" gadget (which is the space bounded computation equivalent of the "alarm-clock" gadget for time bounded computations), and we refer the reader to standard complexity books from the literature for further details. Let us start by sketching the main line of the construction. To start with, we partition $\{0,1\}^*$ in intervals $\{I_i\}_{i\in\mathbb{N}}$ of words of increasing size.

Definition 4.3 *Let $f : \mathbb{N} \to \mathbb{N}$ be defined by $f(0) = 0$ and $f(i+1) = 2^i f(i)^i + 1$.*
$\{I_i\}_{i\in\mathbb{N}}$ is defined by $I_i = \{\omega \in \{0,1\}^* \text{ s.t. } f(i) \leq |\omega| < f(i+1)\}$.

The construction will then be made so as to force the fact that for any $i \in \mathbb{N}$ there exists an element in I_i witnessing the fact that $B \not\leq A$ via R_i. Of course, this will imply that $B \not\leq A$, since the R_i's are an enumeration of reductions. For reasons which will become clear later we need to define a subset of distinguished words D_i inside each one of the intervals I_i.

Definition 4.4 *For every $i \in \mathbb{N}$, D_i is defined as the first i^2 words of I_i (under the canonical ordering of $\{0,1\}^*$). D_i is called the set of* distinguished words *of I_i.*

The whole point of the diagonalization construction we are about to describe is to produce almost complete sets for as many complexity classes as possible. In fact, the construction actually takes, as its main parameter, a complexity class $\mathcal{C}$, but not any complexity class $\mathcal{C}$: in order to have any chance of success, it should be that the complexity class considered fits into a diagonalizable triplet, which definition follows.

Definition 4.5 *A* diagonalizable triplet *is a triplet $(\mathcal{C}, \mathcal{C}, \{\tilde{R}_e\}_{e\in\mathbb{N}})$ such that $\mathcal{C} = \bigcup_{e\in\mathbb{N}}\{L_e\}$ is a complexity class, $\mathcal{C}$ is a complete language for $\mathcal{C}$ (under* logspace *manyone) reductions and $\{\tilde{R}_e\}_{e\in\mathbb{N}}$ is a uniform family of reductions such that:*

1. $\forall e \in \mathbb{N}$ *it either holds that* $L_e \leq \mathcal{C}$ *via* $\tilde{R}_e$ *or that* $\exists d \in \mathbb{N}$ *such that* $L_e = L_d$ *and* $L_d \leq \mathcal{C}$ *via* $\tilde{R}_d$.

2. *Each* $\tilde{R}_e$ *is size increasing:* $\forall e \in \mathbb{N}\ \forall x, y \in \{0,1\}^*$, $|x| \leq |y| \Rightarrow |\tilde{R}_e(x)| \leq |\tilde{R}_e(y)|$

3. $\forall i \neq j\ Im(\tilde{R}_i) \cap Im(\tilde{R}_j) = \emptyset$.

4. $\forall e$ $\tilde{R}_e$ *is one-to-one and there exists an algorithm computing in* $\mathcal{O}(\log(N))$ *space which on input* $y \in Im(\tilde{R}_e)$ *for some* $e \in \mathbb{N}$, *computes the preimage of* y.

5. $\exists \tilde{R}$ *a transducer such that* $\tilde{R}(e, x)$ *computes* $\tilde{R}_e(x)$ *in* $e\log(N) + e$ *space.*

6. *There is a universal algorithm* `Check` *which on input* $(z, j) \in \{0,1\}^* \times \mathbb{N}$ *checks whether* $z \in Im(\tilde{R}_j)$ *in* $\text{poly}(j \cdot \log(|z|))$ *space.*

Convention 4.6 *From now on, we suppose that* $(\mathcal{C}, \mathcal{C}, \{\tilde{R}_e\}_{e\in\mathbb{N}})$ *is a diagonalizable triplet.*

We also need to define a function $g : \mathbb{N} \to \mathbb{N}$, depending on a condition on the distinguished words. Again, we do not try to explain yet the purpose of this function, since we find it easier to do it when it is actually used, in the next subsection.

Definition 4.7 *The function* $g : \mathbb{N} \to \mathbb{N}$ *is defined by* $g(i) = 1$ *if* $\exists x \in D_i$ *such that* $R_i(x) \notin I_i$, $g(i) = 2$ *if* $\exists x_1, x_2 \in D_i$ *such that* $x_1 \neq x_2$ *and* $R_i(x_1) = R_i(x_2)$, $g(i) = 3$ *if* $\exists x \in D_i$ *such that* $R_i(x) \notin \bigcup_{j\leq i} Im(\tilde{R}_j)$ *and* $g(i) = 4$ *otherwise.*

These definitions set the context in which the construction will take place, in subsection 4.2. Before going onto the next stage, we would like to get comfortable with these definitions by proving a few algorithmic facts about them. More precisely, we would like to know that we can efficiently compute the function f of definition 4.3. We would also like to be able to compute, when given a word x, the index i of the interval such that $x \in I_i$ and we would also like to be able to compute the function g. Addressing these questions is what we do next, starting with the following remark which is essentially trivial (and even redundant), but writing these facts down now will avoid us having to justify ourselves later, in the middle or a reasoning where these trivialities may not spring to mind anymore...

Remark 4.8 *Let* $x \in \{0,1\}^N$, *and let* i *be the index such that* $x \in I_i$. *Then:*

1. $2^{2^i} \leq f(i) \leq N \leq f(i+1)$

2. *If* $|x| = f(i)$, *then* $R_i(x) \in \bigcup_{j\leq i} I_j$.

3. *If* x *is a distinguished word of* I_i, *then* $|x| = f(i)$, *therefore* $R_i(D_i) \subseteq \bigcup_{j\leq i} I_j$.

Point 1 holds by solving the first order linear recurrence relation $f(i+1) > f(i)^i$ with initial condition $f(2) > 2$. Point 2 holds because of convention 4.2 and because f was chosen precisely for it to hold while making w.l.o.g. the assumption that the following space-time tradeoff holds: $\texttt{DSPACE}(f(n)) \subseteq \texttt{DTIME}(2^{f(n)})$, 3 is trivial. We now start to investigate the definitions given above from an algorithmic point of view, starting by showing that f is easy to compute.

Lemma 4.9 *There exists an algorithm* `F` *which on input* $i \in \mathbb{N}$ *(encoded in binary) computes* $f(i)$ *in* $\text{polylog}(N)$ *time, where* $N = f(i)$.

Proof. Recall that $f(i) = 2^{i-1}f(i-1)^{i-1} + 1$. Therefore, `F` needs to output $\texttt{BIN}(f(i-1)^{i-1})0^{i-2}1$, where `BIN` is the function mapping a number to its binary encoding. Outputting the $0^{i-2}1$ first bits is easily achieved in polyloglog(N) time, (remembering from remark 4.8 that $i \leq \text{loglog}(N)$). It remains to look how to compute $\texttt{BIN}(f(i-1)^{i-1})$. Let T_i be the run time of `F` on input i. First, `F` makes a recursive call to itself on input $i-1$, and stores the output, $\texttt{BIN}(f(i-1))$, which takes time T_{i-1}. After this, `F` still has to compute $f(i-1)^{i-1}$, which is easily seen to be

computable in polylog(N) time. Thus the following first order linear recursion relation on the T_i's: $T_i = \text{polylog}(N) + T_{i-1}$. Since $i \leq \text{polylog}(N)$ (remark 4.8), solving (or at least bounding) the equation yields $T_i \leq \text{polylog}(N)$. □

The next step in our description of algorithmic facts about the definitions given at the beginning of the section is to make ours an algorithm `Index` which computes, on input x, the index i such that $x \in I_i$. Not surprisingly, the main ingredient is the algorithm `F` of the previous lemma.

Lemma 4.10 *There exists an algorithm* `Index` *which on input* $x \in \{0,1\}^N$ *computes the binary encoding of the index* i *such that* $x \in I_i$*, in* polylog(N) *time.*

Proof. Let $x \in \{0,1\}^N$ be the input to `Index`. First, `Index` computes N the length of its input, which can be done in logarithmic time (by a RAM $\mathfrak{IM}$), c.f. [Bus87]. Then, `Index` repeats:

```
     pseudo-code                 comments
1.   i := 0;                     # Initialize counter
2.   LOOP: repeat {              #
3.     compute f(i+1);           # Using algorithm F from lemma 4.9
4.     if ( f(i+1) > N ){        # Test is true if x ∈ I_i
5.       return i                # Terminate program and return i
6.     }
7.     else {
8.       i := i+1;               # Increment i
9.       next LOOP;              # restart the loop
10.    }
11.  }
```

Clearly this algorithm computes the expected index i such that $x \in I_i$. The only non obvious thing to verify is that the value of $f(i+1)$ in the conditional "if" statement can be computed in polylog(N) time, but this follows from lemma 4.9 which says that $f(i+1)$ is computable in time $\text{polylog}(f(i+1)) = \text{polylog}(2^i f(i)^i) \leq \text{polylog}(N^{\text{loglog}(N)}) = \text{polylog}(N)$ (using remark 4.8). □

Next, we show that we also have an efficient algorithm computing g. While doing so, we shall need the following fact.

Claim 4.11 *Let* ϕ *and* γ *be two polylogarithmic space computable functions, then* $\gamma \circ \phi$ *is polylogarithmic space computable too.*

To substantiate this claim, the idea is to avoid storing the output of ϕ, which could be too large to fit within the announced space bound, but to feed it bitwise to (the algorithm computing) γ. Details are left to the reader.

Lemma 4.12 *There exists an algorithm* `G` *which on input* $i \in \mathbb{N}$ *computes* $g(i)$ *in* polylog(N) *space, where* $N := f(i)$*.*

Proof. We exhibit an algorithm computing g within the announced space bounds. Let i be the input. First use the algorithm `F` of lemma 4.9 to compute and store the (binary encoding of) $f(i) = N$. Next, we would like to construct and store (in a list) the set $D_i = \{x_j\}_{1 \leq j \leq i^2}$. This cannot be done straightforwardly, since $x \in D_i \Rightarrow |x| = f(i)$, which is not polylogarithmic in N. Therefore, instead of storing D_i we store $\bar{D}_i$, a compressed version of D_i which holds on only polylogarithmic[5] in N space: $\bar{D}_i = \{\bar{x}_j\}_{1 \leq j \leq i^2}$, where $\bar{x}_j$ is the word formed by taking the $\log(i^2)$ rightmost bits of x_j[6]. The whole compressed list $\bar{D}_i$ is thus stored on $\mathcal{O}(i^2 \log(i^2))$ bits, which is polylogarithmic in N (remembering remark 4.8).

[5] In fact, $\bar{D}_i$ even hold on polyloglog(N) space, c.f. below.

[6] Recall the D_i are the first i^2 words of length $f(i)$, therefore only the $\log(i^2)$ rightmost bits of an element of D_i can be non zero.

Remark 4.13 *The non compressed version of an element $x \in D$ can be computed in* polylog(N) *space from its compressed version $\bar{x} \in \bar{D}_i$, since $x = 0^{f(i)-|\bar{x}|}\bar{x}$.*

Next, the algorithm tests whether $g(i) = 1$, that is, tests whether $\exists x \in D_i$ such that $|R_i(x)| \leq f(i)$. To do so we compute and compare $f(i)$ (using algorithm `F` of lemma 4.9) and $|R_i(x)|$, with x ranging in D_i. To compute the $|R_i(x)|$'s, with x ranging in D_i, we compute $R_i(\texttt{Dec}(\bar{x}))$, with $\bar{x}$ ranging in $\bar{D}_i$, where `Dec` is a decompression algorithm running in polylog(N) space (c.f. remark 4.13), and count the number of bits outputted (thus avoiding storing the possibly too numerous bits of $R_i(x)$). Claim 4.11 insures that $R_i \circ \texttt{Dec}$ is polylog(N) space computable.

Next, unless $g(i)$ is already known to be equal to 1, test whether $g(i) = 2$. That is, test whether $\exists x_1 \neq x_2 \in D_i$ such that $R_i(x_1) = R_i(x_2)$. We already now that for $x \in D_i$, $R_i(x) = R_i(\texttt{Dec}(\bar{x}))$ can be computed in polylog(N) space (without storing the output). In the previous test, while testing if $g(i) = 1$, we needed to count the length of $R_i(x)$. This time, we need to check whether there exists two distinct elements $x_1, x_2 \in D_i$ with $R_i(x_1) = R_i(x_2)$. To avoid storing the outputs, we make both computations in parallel, and compare the outputs bitwise.

Next, unless $g(i)$'s value is already known, test whether $g(i) = 3$. That is, test whether there exists $y \in D_i$ such that $R_i(y) \in \bigcup_{j \leq i} Im(\tilde{R}_j)$. To be convinced that this step can be carried space efficiently, let us look at one of the tests that have to be made (all of which can be carried out one at a time, after having cleared the memory space previously used). Let $j \leq i$ and $y \in D_i$ be fixed, and suppose it needs to be decided whether $z := R_i(y) \in Im(\tilde{R}_j)$. Through trivial space-time tradeoffs, $|z| \leq 2^{i \log(|y|)+i} < 2^{\mathcal{O}(\log^2(N))}$. Thus, by point 6 of definition 4.5 there exists an algorithm `Check` checking whether $z \in Im(\tilde{R}_j)$ in $\text{poly}(j \cdot \log(|z|)) = \text{poly}(j \log(N)) = \text{polylog}(N)$ space (since $j \leq i \leq \log(N)$. Notice that once again, z may be too long to be stored in memory, so it is fed bitwise (by restarting the whole computation of $R_i(y) = z$ as many times as needed) to `Check`. □

4.2 A Diagonal Construction

In this section, we construct an almost complete set $A_{\mathcal{C}}$. In fact, our construction is parametrizable, once again, so the set constructed will be almost complete only for certain choices of parameters. The main parameter to choose is $(\mathcal{C}, \mathcal{C}, \{\tilde{R}_e\}_{e \in \mathbb{N}})$, a diagonalizable triplet. Here is the main line along which we construct $A_{\mathcal{C}}$: we start by setting $A_{\mathcal{C}} = \mathcal{C}$, i.e. we start with a complete set. Then we modify $A_{\mathcal{C}}$ to break the completeness property, but not too much, so that it remains almost complete. For the time being, we invite the reader to accept the intuitive idea that $A_{\mathcal{C}}$ will be almost complete because we started from a complete problem, and we have only slightly modified it. The detailed and rigorous explanation of the reasons why completeness is almost preserved are delayed until the next section. In this section, we will merely concentrate on the diagonalization process which will braek the completeness of $A_{\mathcal{C}}$. Remember from definition 4.1 that a problem is not complete if there is another problem not reducing to it. We will thus actually construct a problem $B_{\mathcal{C}}$ which does not reduce to $A_{\mathcal{C}}$. The whole difficulty will be to ensure that $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are easy to compute, i.e. that they belong to $\mathcal{C}$. To summarize our discussion, here are the three points we have to fulfill. The third one being discussed only in the next section: 1) $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ should be easy to decide, i.e. it should be that if $\mathcal{C} \in \mathcal{C}$, (for some suitable complexity class $\mathcal{C}$), then $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are in $\mathcal{C}$ too. 2) $A_{\mathcal{C}}$ should *not* be complete, (enforced by the fact that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$). 3) $A_{\mathcal{C}}$ should be almost complete, i.e. most of the languages reducing to $\mathcal{C}$ should reduce to $A_{\mathcal{C}}$ too.

As we mentioned before, the construction is done by diagonalizing against $\{R_i\}_{i \in \mathbb{N}}$, the effective enumeration of reductions from convention 4.2. We start with $A_{\mathcal{C}} := \mathcal{C}$ a complete language, and $B_{\mathcal{C}} := \emptyset$ the empty language. Then, at step i for $i \in \mathbb{N}$ we modify $A_{\mathcal{C}} \cap I_i$ or $B_{\mathcal{C}} \cap I_i$ to ensure that there is an element of I_i witnessing the fact that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i, i.e. we ensure that there exists $x \in I_i$ such that $x \in B_{\mathcal{C}} \iff R_i(x) \notin A_{\mathcal{C}}$. The exact definition is as follows.

Definition 4.14 *Let $\mathcal{C}$ be a language. The languages $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are defined recursively on the intervals of the partition I (from definition 4.3). For the initial step: $A_{\mathcal{C}} \cap I_0 = \mathcal{C} \cap I_0$ and*

$B_{\mathcal{C}} \cap I_0 = \emptyset$. Then by induction, if $A_{\mathcal{C}} \cap I_j$ and $B_{\mathcal{C}} \cap I_j$ are defined $\forall j < i$, $A_{\mathcal{C}} \cap I_i$ and $B_{\mathcal{C}} \cap I_i$ are defined according to one of the four following cases, depending on the value of the function g of definition 4.7:

(1) If $g(i) = 1$. *In this case, we leave $A_{\mathcal{C}}$ unchanged, $A_{\mathcal{C}} \cap I_i := \mathcal{C} \cap I_i$. By case assumption, there exists $\xi := \min\{x \in D_i | R_i(x) \notin I_i\}$ and we use this fact to insure that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i, by letting x be such that $R_i(x) = \xi$ and by setting $B_{\mathcal{C}} \cap I_i = \emptyset$ in the case where $\xi \in A_{\mathcal{C}}$, and by setting $B_{\mathcal{C}} \cap I_i = \{x\}$ if $\xi \notin A_{\mathcal{C}}$. (Notice that since $g(i) = 1$ and $\xi = R_i(x)$ with $x \in D_i$, then $\xi \in \bigcup_{j<i} I_j$ (c.f. point 3 of remark 4.8), and thus the question whether $\xi \in A_{\mathcal{C}}$ or not is answered in a previous stage of the construction).*

(2) If $g(i) = 2$. *$A_{\mathcal{C}}$ is left unchanged again, $A_{\mathcal{C}} \cap I_i := \mathcal{C} \cap I_i$. Let $\xi_1 < \xi_2$ be the two smallest elements (which exist by case assumption) of D_i such that $R_i(\xi_1) = R_i(\xi_2)$. We set $B_{\mathcal{C}} \cap I_i = \xi_1$, which insures that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i since either $\xi_1 \in B_{\mathcal{C}}$ and $R_i(\xi_1) \notin A_{\mathcal{C}}$ or $\xi_2 \notin B_{\mathcal{C}}$ and $R_i(\xi_2) \in A_{\mathcal{C}}$.*

(3) If $g(i) = 3$. *Let $\xi := \min\{x \in D_i | R_i(x) \notin \bigcup_{j \leq i} Im(\tilde{R}_j)\}$. This time we have to modify $A_{\mathcal{C}}$, by setting $A_{\mathcal{C}} \cap I_i = \mathcal{C} \cap I_i \cup \{R_i(\xi)\}$. We let $B_{\mathcal{C}} \cap I_i = \emptyset$ so that it holds that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i since $\xi \notin B_{\mathcal{C}}$ but $R_i(\xi) \in A_{\mathcal{C}}$.*

(4) If $g(i) = 4$. *By case assumption, $R_i(D_i) \subseteq \bigcup_{j \leq i} Im(\tilde{R}_i) \cap I_i$, and $R_{i_{|D_i}}$ is one-to-one, since otherwise $g(i) \leq 2$. Since $|D_i| = i^2$, the pigeon hole principle implies that there exists $e_i := \min\{1 \leq l \leq i \ : \ |R_i(D_i) \cap Im(\tilde{R}_l)| \geq i\}$. Let J_i be the first i elements of D_i that are mapped by R_i to $Im(\tilde{R}_{e_i})$, and let $F_i := R_i(J_i)$. If $\mathcal{C} \cap F_i = \emptyset$, then let $\xi := \max F_i$. We change $A_{\mathcal{C}}$ by setting $A_{\mathcal{C}} \cap I_i = \mathcal{C} \cap I_i \cup \{\xi\}$ while $B_{\mathcal{C}}(x) \cap I_i = \emptyset$. Notice that if $\mathcal{C} \cap F_i \neq \emptyset$, then $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i, (and $B_{\mathcal{C}} \not\leq \mathcal{C}$ via R_i either) since $\exists x \in J_i$ such that $R_i(x) \in A_{\mathcal{C}} \cap \mathcal{C}$ but $x \notin B_{\mathcal{C}}$. In the case where $\mathcal{C} \cap F_i \neq \emptyset$, ξ ensures that $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i.*

As emphasised throughout the previous discussion, this definition guarantees that the following lemma holds

Lemma 4.15 *$\forall i \in \mathbb{N}$ $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ via R_i, and thus $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$.*

The proof of it is the fact that by construction the following holds: $\forall i \in \mathbb{N}$ $\exists x \in I_i$ such that $[x \in B_{\mathcal{C}} \iff R_i(x) \notin A_{\mathcal{C}}]$. The next step for us is now to prove that $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are efficiently decidable if $\mathcal{C}$ is. Thus $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ will be in $\mathcal{C}$, at least for some settings of the parameters. Before we accomplish this, there is another technical result we need. This technical result is similar to the ones from the previous section in spirit, which gave us knowledge on the algorithmic structure of the intervals $\{I_i\}_{i \in \mathbb{N}}$, only this time, the results will concern not the I_i's, but the sets D_i, J_i and F_i. (The former two having been defined only in definition 4.14, so it would not have been possible to include these results in the previous section). Roughly speaking, what we would like to prove is that it is possible to compute and store these sets, and that the computation space required plus the space required to store the output is small. Actually, this is not the case, since the sets themselves are large (so is said informally). Nevertheless, the information content of these sets is in some sense small, so we show that it is possible to *pretend* that these sets are computed and stored in such a way that a subroutine needing to access them could do so transparently, without noticing any difference with the situation in which they would effectively be stored. The idea in order to do so is standard: store a space efficiently computable *compressed form* of these sets, and make use of a space efficient decompression algorithm to serve them transparently to any subroutine needing to access them. This sums up to the following definitions.

Definition 4.16 *Let $X = \{x_i\}_{1 \leq i \leq m}$ be a set of m words. The* storage size *of X is defined as $X^\sharp = \sum_{i=1}^{m} |x_i|$.*

Definition 4.17 (Compressed forms) *Let $\{X_i\}_{i \in \mathbb{N}}$ be a family (of finite languages) with $X_i = \{x_{i,j}\}_{1 \leq j \leq k_i}$ a finite set of words. We say that $\{X_i\}_{i \in \mathbb{N}}$ can be* compressed on $s(i)$ bits *if there*

exists a family $\{\bar{X}_i\}_{i\in\mathbb{N}}$ *with* $\bar{X}_i = \{\bar{x}_{i,j}\}_{1\leq j\leq k_i}$ *a finite set of words and a uniform family of decompression algorithms* $\{\texttt{Dec}_i\}_{i\in\mathbb{N}}$ *such that* $\texttt{Dec}_i(\bar{x}_{i,j})$ *computes* $x_{i,j}$ *in* $s'(i)$ *space, with* $s(i) = X_i^\sharp + s'(i)$. *We call* $\bar{X}_i$ *the* compressed form *of* X_i *and* $\bar{x}_{i,j}$ *the* compressed representative *of* $x_{i,j}$.

The idea is that if a set X needs to be stored it is possible to store $\bar{X}$ instead, and through the use of Dec, grant a subroutine access to X in a transparent way. It may be noticed that this may not be time efficient, since Dec may be called many times, but it is space efficient whenever X can be compressed on far less bits then would be required to store X uncompressed. The attentive reader may also have noticed that in our previous discussion, we addressed the problem of not only storing and decompressing space efficiently a set, but we also said that it would be reasonable to be able to space efficiently compute the compressed forms of the set we want to store, and that no reflexion of this fact appears in definition 4.17. This is a deliberate choice, motivated by the will to enhance comprehensiveness in giving later the definition 4.20 of *effective compressions* which tackles this issue. We take the following convention.

Convention 4.18 *When we say that* "X_i can be compressed on $s(i)$ bits and decompressed via the algorithm Dec" *, it is implicit that we are in fact considering a family of finite languages* $\{X_i\}_{i\in\mathbb{N}}$ *and a uniform family of algorithms* $\{\texttt{Dec}_i\}_{i\in\mathbb{N}}$, *as in the previous definition.*

We start by giving the following compression result on the set of distinguished words D_i and the related sets J_i and F_i.

Lemma 4.19 D_i, J_i *and* F_i *can be compressed on* polylog$(f(i))$ *bits.*

Proof. By definition $D_i = \{x_j\}_{1\leq j\leq i^2}$ is the set of i^2 first words of I_i. Therefore $x_1 = 0^{f(i)}$, $x_2 = 0^{f(i)-1}1$, $x_3 = 0^{f(i)-2}10$, $x_4 = 0^{f(i)-2}11$, etc..., Since $|D_i| = i^2$, each of the x_j's in D_i have all their bits equal to 0, except for (some of) their $\log(i^2)$ rightmost bits. For each $1 \leq j \leq i^2$, we consider $\bar{x}_j$ the compressed representative of x_j, which is defined as the i^2 rightmost bits of x_i, and $\bar{D}_i := \{\bar{x}_j\}_{1\leq j\leq i^2}$ the compressed form of D_i, with storage size $\bar{D}_i^\sharp = i^2\log(i^2)$. The decompression algorithm $\texttt{Dec}(\bar{x}_j)$ only needs to pad $f(i)-\log(i^2)$ leading 0's on the left of $\bar{x}_j$. The computational space required by Dec is essentially the space required to compute $f(i)$. By lemma 4.9, $f(i)$ is computable in polylog$(f(i))$ time and space. We have thus shown that D_i can be compressed on $\bar{D}_i^\sharp$+polylog$(f(i))$ = polylog$(f(i))$ bits. It is now trivial that J_i can be compressed on polylog$(f(i))$ bits, since by definition $J_i \subseteq D_i$, we have $\bar{J}_i \subseteq \bar{D}_i$ and the decompression algorithm is the same. (Notice that we have made no claim on how to compute $\bar{J}_i$, but only that this set exists). Since $F_i = R_i(J_i)$, we take $\bar{F}_i = \bar{J}_i$, but if Dec is the decompression algorithm for J_i (and D_i), the decompression algorithm for F_i is $R_i \circ \texttt{Dec}$. The space required to compute $R_i \circ \texttt{Dec}$ is the space to compute Dec, which we have already seen to be polylog$(f(i))$, plus the space to compute R_i on elements of D_i, which is $i\log(f(i)) + i$ = polylog$(f(i))$ as follows from the definition of R_i and since $D_i \subseteq \{0,1\}^{f(i)}$, claim 4.11 permits to finish the proof. □

This result in its current state is not yet useful: as previously suggested, we would like the same result, but with the fact that the compressed forms are space efficiently computable too, thus the motivation of the next definition and lemma.

Definition 4.20 *We say that a set* X_i *can be* effectively compressed *on* $s(i)$ *bits if it can be compressed on* $s(i)$ *bits and if, on input* i, *its compressed form* $\bar{X}_i$ *can be constructed using* $s'(i)$ *working space with* $s(i) = \bar{X}_i^\sharp + s'(i)$.

Lemma 4.21 D_i, J_i *and* F_i *can be* effectively compressed *on* polylog$(f(i))$ *bits.*

Proof. From lemma 4.19, we already know that these sets can be compressed within the announced space. It only remains to show that their compressed forms, $\bar{D}_i$, $\bar{J}_i$ and $\bar{F}_i$ can be constructed in polylog$(f(i))$ space too. The case of $\bar{D}_i$ is easy: compute $\bar{x}_1 = 0^{\log(i^2)}$, $\bar{x}_2$ = "$\bar{x}_1 + 1$", etc... (Details are left to the reader). We now look at the case of J_i. First of all, recall from definition

4.14 that J_i is only defined for i such that $g(i) = 4$. So we first compute $g(i)$, and unless $g(i) = 4$ we have $J_i = \emptyset$. Computing $g(i)$ takes polylog$(f(i))$ space (lemma 4.12). If $J_i \neq \emptyset$, i.e. if $g(i) = 4$, we compute e_i (as defined under point 4 of definition 4.14). To compute $e_i = \min\{l \in \{1, ..., i\} : |R_i(D_i) \cap Im(\tilde{R}_l)| \geq i\}$, apart from simple counting, what has to be done is to check whether $R_i(y) \in Im(\tilde{R}_j)$ for y's in D_i and $j \leq i$. We already know from the proof of lemma 4.12 that this check can be done in polylog$(f(i))$ space. We can suppose that we have at our disposal the element of D_i stored in memory, since we have already shown that D_i can be effectively compressed on poly$(f(i))$ space. In fact, we can also suppose that we have $R_i(D_i)$ stored in memory, since it is easy to see that $R_i(D_i)$ can be effectively compressed on poly$(f(i))$ space: indeed, we set $\overline{R_i(D_i)} = \bar{D}_i$, and the decompression algorithm is the same as the decompression algorithm for $\bar{J}_i$ (details are left to the reader). Now to enumerate the elements of $\bar{J}_i$ we only need to check, for each element $x_i \in R_i(D_i)$ if it belongs to $Im(\tilde{R}_{e_i})$. If this is so, the representative $\bar{x}_i$ of x_i belongsx to $\bar{J}_i$, otherwise it does not. The check is done via the algorithm `Check` which exists by point 6 of definition 4.5, which runs in poly$(e_i \log(|R_i(D_i)|)) \leq$ poly$(i \log(2^{i \log(f(i))+i}))$ = poly$(f(i))$ space. □

We are now, at last, equipped with the adequate tools to prove that $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are efficiently decidable when $\mathcal{C}$ is. Since $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are defined on each interval, applying a different rule depending on the value of g on the index of this interval, the first step in constructing an algorithm deciding $A_{\mathcal{C}}$ (or $B_{\mathcal{C}}$) is to have a subroutine capable, on input $x \in \{0,1\}^*$, of finding the index i such that $x \in I_i$, and another computing the value of $g(i)$. These subroutines are already available through the algorithms `Index` and `G` (lemmas 4.10 and 4.12). To decide $A_{\mathcal{C}}$ or $B_{\mathcal{C}}$, it only remains to consider the computations naturally arising from definition 4.14.

Theorem 4.22 *If $\mathcal{C}$ is decidable in $s(N)$ space and $t(N)$ time, then $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are decidable in $\mathcal{O}(\max\{s(N)\}, \text{polylog}(N))$ space and $\mathcal{O}(\log(N)t(N) + 2^{\text{polylog}(N)})$ time.*

Proof. We give an algorithm deciding $A_{\mathcal{C}}$ within desired space bounds. Let $x \in \{0,1\}^N$. We need to output $A_{\mathcal{C}}(x)$. First, compute i, $g(i)$ and $f(i)$ such that $x \in I_i$, (from lemmas 4.9, 4.10 and 4.12, this can be done in polylog(N) space).

If $g(i) = 1$ or $g(i) = 2$, then output $\mathcal{C}(x)$ which can by hypothesis be either computed in $t(N)$ time or in $s(N)$ space.

Else, if $g(i) = 3$, find ξ the smallest element of D_i such that $R_i(\xi) \notin \bigcup_{j \leq i} Im(\tilde{R}_j)$. This can be done in polylog(N) space. The arguments needed to substantiate this assertion are identical to the ones from lemma 4.12 or 4.21, and left to the reader[7] . If $R_i(\xi) = x$, (this test must once again be done bitwise in order to avoid storing all the output), output 1, else output $\complement(x)$.

Finally, if $g(i) = 4$, then compute e_i in polylog(N) space (as in lemma 4.19). Since D_i, J_i and F_i can be effectively compressed (lemma 4.21) , we can w.l.o.g. suppose that they are constructed, stored and available for access, and that all this requires polylog(N) space. We can thus w.l.o.g. suppose that D_i, J_i and F_i are stored and available in memory, at an "additional" cost of polylog(N) space,

Next, find $\bar{y}$ the compressed representative of $y := \max F_i$. This can be done by (a sorting algorithm) comparing the elements of J_i bitwise, thus never needing to actually store in memory a decompressed representative of an element of F_i. If $x \neq y$, we output $\mathcal{C}(x)$. If $x = y$, it depends: if $\mathcal{C} \cap F_i = \emptyset$, we output 1, else we output $\mathcal{C}(x)$. This implies we need to test whether $\mathcal{C} \cap F_i = \emptyset$. The elements of F_i are accessible via their compressed enumeration $\bar{F}_i$, and each member of F_i is of size at most N, since $N = |x|$ and $x = \max(F_i)$ by case assumption. Thus for each of the at most $\log(N)$ y's in F_i, we can check if $y \in F_i$ in $t(N)$ time and $s(N)$ space.

From the definition of $A_{\mathcal{C}}$, it is clear that this algorithm decides $A_{\mathcal{C}}$, and putting together the complexity analysis from above, one sees that the computation requires $\mathcal{O}(\max\{\text{polylog}(N), s(N)\})$ space and $\mathcal{O}(\log(N)t(N) + 2^{\text{polylog}(N)})$ time. A similar proof shows that the same holds for $B_{\mathcal{C}}$. □

[7] Fortunately, ξ does not need to be stored. We can take advantage of lemma 4.21 to store the compressed representative of ξ on polylog(N) space.

4.3 Almost Completeness

In the previous section, we constructed the language $A_{\mathcal{C}}$, which we propose as a candidate almost complete problem. Roughly summarizing, we already know that $A_{\mathcal{C}}$ is easy to decide, in the sense that for a suitable complexity class $\mathcal{C}$, lemma 4.22 implies that $A_{\mathcal{C}} \in \mathcal{C}$, and that $A_{\mathcal{C}}$ is not complete, since $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$, (lemma 4.15). We have not given any idea of why $A_{\mathcal{C}}$ should be almost complete, besides the intuitive idea that it is a language which is constructed by starting from a complete problem $\mathcal{C}$, and since we modified it only a bit, probably something of this completeness remains, and it seems plausible that $A_{\mathcal{C}}$ would be almost complete. To prove formally that $A_{\mathcal{C}}$ is almost complete, we have to prove that many languages of $\mathcal{C}$ reduce to $A_{\mathcal{C}}$. This means that we need to prove that the lower span of $A_{\mathcal{C}}$ is large, or equivalently, that the complementary of the span is small, in terms of an $\mathcal{RBM}$. Of course this will not be true for any $\mathcal{RBM}$, but we would like it to be true for the type of $\mathcal{RBM}$'s such as the ones we defined in section 3.3 in terms of martingales. Therefore, we shall exhibit a martingale which covers every language of $\mathcal{C}$ not reducing to $A_{\mathcal{C}}$, (thus intuitively showing that the complementary of the span of $A_{\mathcal{C}}$ is small) and which is computable using little computing resource. This potentially implies that the span of $A_{\mathcal{C}}$ is large for many complexity classes. We start by describing the martingale, the coverage property will then be proved in theorem 4.27 and the efficient computation property in theorem 4.34. The martingale we need to describe, having the here above cited properties is, not surprisingly, constructed in close relation with the diagonalization construction of $A_{\mathcal{C}}$ described in the previous section. Therefore, let us start with the following convention.

Convention 4.23 *$\forall i \in \mathbb{N}$ such that $g(i) = 4$, the integer $e_i \in \mathbb{N}$, the sets J_i and F_i are defined as in the case 4 of definition 4.14. If $g(i) \neq 4$, we do not define e_i nor J_i, and we let $F_i := \emptyset$.*

The description of d, a martingale covering the complement in $\mathcal{C}$ of $P_m^{\leq}(A_{\mathcal{C}})$ will be, *grosso modo*, split into three phases. First, we will describe d as a pure mathematical function, but in an intuitive way, through the use of the casino game. In a second stage, we will describe the success set $S^{\infty}[d]$ of d, and finally, we will describe and analyse the complexity of an algorithm computing d. We start with the formal description of the martingale d. We do not describe d directly, but describe a family of martingales $\{\beta_e\}_{e\in\mathbb{N}^*}$ instead, which can be assembled to form d by summing up the β_e's. How this is done is explained later in this section. Actually, we do not really describe any of the β_e's either. Instead, for each fixed $e \in \mathbb{N}$, we describe a family $\{\beta_{e,i}\}_{i\in\mathbb{N}^*}$ of martingales, and define $\beta_e = \sum_{i\in\mathbb{N}} 2^{-3i}\beta_{e,i}$. This trick is standard, and we leave it to the reader to verify that if each of the $\beta_{e,i}$'s are martingales, then so is β_e. Notice that computing β_e will most probably be very difficult since it implies an infinite sum. We shall address this problem later, and for the time being, we are only interested in defining the somehow "purely mathematical" (as opposed to computable) family of martingales β_e. In our case, each of the martingales in $\{\beta_{e,i}\}_{i\in\mathbb{N}^*}$ wagers only a finite number of times, and no two different martingales in this family wager on a same word. In some sense, the idea is that the set of words is partitioned into subsets of words, and β_e delegates the charge of wagering (while playing the casino game) on the ith subset to the ith martingale $\beta_{e,i}$. Since each of the $\beta_{e,i}$ only wagers a finite number of times, their total capital is always necessarily bounded, so each of the $\beta_{e,i}$ will have an empty success set $S^{\infty}[\beta_{e,i}]$, but this is not necessarily true for β_e, since the infinite sum $\sum_{i\in\mathbb{N}^*} 2^{-3i}\beta_{e,i}$ may of course not be bounded, even if each term in the sum is.

Definition 4.24 *$\forall e \in \mathbb{N}$, let $W_{e,i} = \tilde{R}_e^{-1}(F_i)$, and $W_{e,i}^{\leq x} = W_{e,i} \cap \{y \in \{0,1\}^* | y \leq x\}$.*

Definition 4.25 *Fix $e \in \mathbb{N}$. Consider β_e the following martingale to play the casino game. The first step is to split the initial martingale β_e in a family of martingales $\{\beta_{e,i}\}_{i\in\mathbb{N}^*}$ by letting $\beta_e = \sum_{i\in\mathbb{N}^*} 2^{-3i}\beta_{e,i}$. The ith martingale $\beta_{e,i}$ is used to bet on words of $W_{e,i}$ only, so each of the $\beta_{e,i}$'s wagers on a distinct subset of words than the other $\beta_{e,i}$'s, since $W_{e,i} \cap W_{e,j} = \emptyset$ if $i \neq j$. Suppose the casino has chosen an initial language L. Let us describe the strategy of β_e when it comes to bet on the Nth bit of χ_L. In other words, we need to describe the value of $\beta_e(\omega[N])$, where $\omega = \chi_L$. Since $\beta_e = \sum_i 2^{-3i}\beta_{e,i}$, this is done by describing the family of martingales $\{\beta_{e,i}\}_{i\in\mathbb{N}^*}$:*

1. *Unless $\exists i$ such that $s_N \in W_{e,i}$, do not wager: none of the $\beta_{e,i}$ changes, and so $\beta_e(\omega[N]) = \beta_e(\omega[N-1])$.*

2. *Otherwise, if $s_N \in W_{e,i}$ (for some $i \in \mathbb{N}$), only the ith martingale $\beta_{e,i}$ changes, playing according to the following strategy: bet the whole capital $\beta_{e,i}(\omega[N-1])$ on the fact that $s_N \notin L$, i.e. the wager is placed on $\omega(N) = 0$, thus doubling it if $\omega(N) = 0$ and loosing it all if $\omega(N) = 1$: $\beta_{e,i}(\omega[N]) = 2(1-\omega(N))\beta_{e,i}(\omega[N-1])$.*

We leave it to the reader to verify that each of the $\beta_{e,i}$ is a martingale, which implies in turn that β_e is a martingale too.

Convention 4.26 *The martingale β_e described here above depends on e. The family $\{\beta_e\}_{e\in\mathbb{N}}$ is the family of martingales obtained by letting e range over $\mathbb{N}$, and we refer to β_e as the eth martingale.*

Lemma 4.27 ([ASMRT03]) *Let $e \in \mathbb{N}$ be an integer such that L_e does not reduce to $A_{\mathcal{C}}$, then there are infinitely many i's in $\mathbb{N}$ such that $g(i) = 4$ and $e_i = e$ and $W_{i,e} \cap L_e = \emptyset$.*

The fact that this lemma holds comes from the structure induced by the diagonalization process of definition 4.14.
Proof. Suppose that $L_e \not\leq$ to $A_{\mathcal{C}}$. Then, necessarily, it does not reduce to $A_{\mathcal{C}}$ via a finite variation of $\tilde{R}_e$. This implies that there are infinitely many words x such that $\tilde{R}_e(x) \in A_{\mathcal{C}} \iff \tilde{R}_e(x) \notin \mathcal{C}$, since L_e reduces to $\mathcal{C}$ (via $\tilde{R}_e$), and *not* to $A_{\mathcal{C}}$. By analysing the construction of $A_{\mathcal{C}}$ (definition 4.14), we see that this implies that [$g(i) = 4$ and $e_i = e$ and $\mathcal{C} \cap F_i = \emptyset$] for infinitely many i's in $\mathbb{N}$, (detailed explanation of this latter fact can be found in [ASMRT03]). We then notice that for those infinitely many i's, the following holds. $\mathcal{C} \cap F_i = \emptyset \Rightarrow \tilde{R}_{e_i}^{-1}(\mathcal{C}) \cap \tilde{R}_{e_i}^{-1}(F_i) = \emptyset \Rightarrow L_e \cap W_{i,e} = \emptyset$, which finishes the proof. □

With this lemma, we can state the following theorem.

Theorem 4.28 ([ASMRT03]) *If L_e does not reduce to $A_{\mathcal{C}}$, then $L_e \in S^{\infty}[\beta_e]$.*

Proof. Let $e \in \mathbb{N}$ be a fixed integer. Suppose that $L_e \not\leq A_{\mathcal{C}}$. By theorem 4.27, there are thus infinitely many i's such that $g(i) = 4$, $e_i = e$ and $W_{e,i} \cap L_e = \emptyset$. Let us look at what happens to the ith martingale $\beta_{e,i}$ for one of these infinitely many i's, while playing against the language L_e. By definition, $\beta_{e,i}$ is used to wager on words of $W_{e,i}$, and sooner or later, it will have been used to play on every word of $W_{e,i}$. The strategy is to always play the whole current capital, thus, each time $\beta_{e,i}$ wagers, it either doubles or quits. In the end, $\beta_{e,i}$ will either have been doubled $|W_{e,i}|$ times, reaching the peak value of $2^{|W_{e,i}|}$ (this happens if $L \cap W_{e,i} = \emptyset$), or completely lost (if $L \cap W_{e,i} \neq \emptyset$). Since by assumption we are looking at an i such that $L_e \cap W_{e,i} = \emptyset$, this means that $\beta_{e,i}$ reaches the peak value of $2^{|W_{e,i}|}$.

Remark 4.29 *$|W_{e,i}| = i$ if $e = e_i$ and $g(i) = 4$, $|W_{e,i}| = 0$ otherwise.*

This remark is seen to be true by remembering that by definition, $F_i \subseteq Im(\tilde{R}_{e_i})$ (point 4 of definition 4.14), that $Im(\tilde{R}_i) \cap Im(\tilde{R}_j) = \emptyset$ if $i \neq j$ (point 4 of convention 4.5), that $|F_i| = i$ and that the $\tilde{R}_i$'s are one-to-one. We now turn back to β_e: since $\beta_e = \sum_{i\in\mathbb{N}^*} 2^{-3i}\beta_{e,i}$, it means that in the end, each of the infinitely many indexes i such that $g(i) = 4$, $e_i = e$ and $W_{e,i} \cap L_e = \emptyset$ will contribute to the value of $\beta_e = \sum_{i\in\mathbb{N}}$ by $2^{|W_{e,i}|}/2^i = 1$, (as follows from remark 4.29). Therefore, when $N \to \infty$, there are infinitely many terms in the sum above reaching a peak value of 1. Thus $\lim_{N\mapsto\infty} \beta_e(L[N]) = \infty$, and thus, by definition, $L \in S^{\infty}[\beta_e]$. □

This last theorem gives an important covering property of the family of martingales $\{\beta_e\}_{e\in\mathbb{N}^*}$. In order to make use of this property, we are to show that they can also be computed in an efficient way. In fact, we will also need to show that the union of their success set is covered by a single efficiently computable martingale. We start by describing a uniform algorithm computing the β_e's.

Lemma 4.30 *There exists a* $\Gamma(\mathcal{O}(e \log\log N))$ $\mathfrak{TM}$ $T^{|x|}$ *such that* $T^{|x|}(e, i, \omega[N]) = \beta_{e,i}(\omega[N])$.

Proof. Suppose we want to compute $\beta_{e,i}(\omega[N])$ for some $\omega \in \{0,1\}^\infty$ and some $N \in \mathbb{N}$. Let s_N be the Nth word and $n = \log(N) \simeq |s_N|$. Let us define the following:

Claim 4.31 *The algorithm below computes* $\beta_{e,i}(w[N])$.

```
1.  Compute s_N;
2.  Compute j := Index( z = R̃_e(s_N) );
3.  If ( j < i ) return 1;
4.  Else{
5.      Unless ((g(i) = 4) and (e_i = e)) return 1;
6.      Compute W_{e,i} and W_{e,i}^{≤s_N};
7.      result := 1;
8.      foreach ({k ∈ ℕ | s_k ∈ W_{e,i}^{≤s_N}}){
9.          if (ω[k] ≠ 0) return 0;
10.         else result := result * 2;
11.     }
12.   return result;
13. }
```

First of all, we substantiate the claim. Remember that the martingale $\beta_{e,i}$ is only modified when the strategy β_e wagers on a word belonging to $W_{e,i}$. Point 3: First, notice that for $\beta_{e,i}$ to be modified at the kth stage of the game, i.e. for $\beta_{e,i}(\omega[k])$ to be different from $\beta_{e,i}(\omega[k-1])$, it has to be that $s_k \in W_{e,i}$, which implies that $\tilde{R}_e(s_k) \in I_i$. Next, notice that, since $\tilde{R}_e$ is size increasing (c.f. point 2 of definition 4.5), if $\texttt{Index}(\tilde{R}_e(s_N)) < i$, then $\forall k < N$ $\texttt{Index}(\tilde{R}_e(s_k)) < i$, and therefore $\forall k \leq N$ $s_k \notin W_{e,i}$. Having noticed these two points, we see that if $\texttt{Index}(\tilde{R}_e(s_N)) < i$, then $\beta_{e,i}(\omega[N]) = \beta_{e,i}(\omega[0])) = 1$, thus the correctness of point 3. Point 5: unless $g(i) = 4$ and $e_i = e$, we have $W_{e,i} = \emptyset$, (c.f. definitions 4.14 and 4.24), and the martingale $\beta_{e,i}$ thus never wagers, so $\beta_{e,i}$ remains forever unchanged from its initial value of 1. Unless this easy case occurs, $\beta_{e,i}$ wagers on the words of (the non-empty set) $W_{e,i}$, betting on the fact that $\omega[k] = 0$ for every k such that $s_k \in W_{e,i}$, each time either doubling its capital or losing it all. The "for" loop of point 7 ensures that the algorithm outputs $2^{|W_{e,i}^{\leq s_N}|}$ if $L_\omega \cap W_{e,i}^{\leq s_N} = \emptyset$, and 0 otherwise.

Remark 4.32 *If the points* 2 *and* 3 *where removed in the above algorithm, the output would remain the same, because if* $\texttt{Index}(\tilde{R}_e(s_N)) < i$, *then* $W_{e,i}^{\leq s_N} = \emptyset$ *and point* 7 *would return* 1 *anyway. The purpose of points* 2 *and* 3 *is to ensure that the complexity of the algorithm does not depend on the value of* i, *by avoiding point* 6 *and the computation of* $W_{e,i}$, *when it does not hold that* $i << N$, *(more on this below).*

Next we analyse the space complexity required to run this algorithm. For point 1, remember that s_N is by convention already available on an auxiliary input tape, so it does not really need to be computed, and although its storage size is $\mathcal{O}(\log(N))$, we do not count it in the space complexity analysis of our algorithm, since we will never have to transfer it on a working tape. Point 2: by definition 4.5, $\tilde{R}_e(s_N)$ is computable in $e\log(|s_N|) + e = \mathcal{O}(e \log\log(N))$ space. Through trivial space-time tradeoffs, it implies that $|\tilde{R}_e(s_N)| \leq 2^{\mathcal{O}(e\log\log(N))} = \text{poly}(\log^e(N))$ (note that $\tilde{R}_e(s_N)$ does not need to be stored, it only needs to be fed bitwise to `Index`; i.e. it suffices to store $\tilde{R}_e(s_N)$ in a *compressed form*, c.f. definition 4.19). By lemma 4.10 and (arguments similar to the ones in) lemma 4.11, it thus holds that $\texttt{Index}(\tilde{R}_e(s_N))$ is computable in $\text{polylog}(\text{poly}(\log^e(N))) = \text{poly}(e\log\log(N))$ time and space. Storing j is no problem, since by remark 4.8 it holds that $j \leq \log\log(|\tilde{R}_e(s_N)|) \leq \log\log \text{polylog}^e(N)$. Summarizing the above steps, we see that points 2 and 3 are computable in $\text{poly}(e\log\log(N))$. Point 5: from lemma 4.12, we can test if $g(i) = 4$ in $\text{polylog}(f(i))$ space, and as in theorem 4.22 and lemma 4.12, we can test if $e_i = e$ in $\text{polylog}(f(i))$ space too, with $f(i) \leq \text{poly}(\log^e(N))$. This last inequality holds because (and it is now that points

2 and 3 in the algorithm become important) we are guaranteed that $\texttt{Index}(\tilde{R}_e(s_N)) \geq i$, and thus $\tilde{R}_e(s_N) \in I_i$, which implies, by definition of I_i, that $f(i) \leq |\tilde{R}_e(s_N)| \leq \text{poly}(\log^e(N))$. Therefore, point 5 is computable in space $\text{polylog}(f(i)) \leq \text{polylog}(\text{poly}(\log^e(N))) = \text{poly}(e \log\log(N))$. Point 6: we know from lemma 4.21 that F_i can be effectively compressed on $\text{polylog}(f(i))$ space. Since $W_{e,i} = \tilde{R}_e^{-1}(F_i)$, this holds for $W_{e,i}$ too (c.f. point 4 of definition 4.5). Since we also have access to s_N, $W_{e,i}^{\leq s_N}$ can thus trivially also be effectively compressed on $\text{polylog}(f(i))$ space. Thus point 6 is computable in $\text{polylog}(f(i)) \leq \text{poly}(e \log\log(N))$ space. The bits of $\omega[N]$ queried in the "if" statement of the "for" loop are those with indexes corresponding to words of $W_{e,i}^{\leq s_N}$. To query such a bit, words of $W_{e,i}^{\leq s_N}$ have to be decompressed and written on the query tape. These decompressed words could be too large to fit in memory, if they were to be decompressed on a work tape and then copied to the query tape. We avoid this problem by writing the output of the decompressing algorithm directly to the query tape, which is not taken into account while analyzing the space complexity of the algorithm (convention 3.2). Also, since the algorithm queries its input in $W_{e,i}^{\leq s_N}$ we can be sure that it queries, for any input, in a set $G_{e,i} := W_{e,i}$, which is thus of cardinality at most i, with $2^{2^i} < f(i) \leq 2^{e \log(|s_N|)+e}$, and thus $|G_{e,i}| < \log(e) + \log\log\log(N) = \mathcal{O}(\log(e \log\log(N)))$. The size of the set of queried words is thus even logarithmically smaller than what was announced in the statement of the lemma. □

We can now combine the two previous results, theorem 4.28 and lemma 4.30, to show that there is a unique martingale d which is easy to compute and which covers the union of the success sets of the β_e's, thus it covers any language of $\mathcal{C}$ not reducing to $A_{\mathcal{C}}$.

Lemma 4.33 *There exists d a $\Gamma(\text{polylog}\log(N))$ martingale such that $\bigcup_{e\in\mathbb{N}} S^\infty[\beta_e] \subseteq S^\infty[d]$.*

Proof. By definition 4.25, the eth martingale β_e is equal to $\sum_{i\in\mathbb{N}^*} 2^{-3i}\beta_{e,i}$. Let us start by showing that each β_e can be efficiently approximated. Let $\tilde{\beta}_e(\omega[N]) = \sum_{i=1}^{\log(N)} 2^{-3i}\beta_{e,i}(\omega[N])$. $\tilde{\beta}_e$ is an $1/N^2$ close approximation of β_e: $\beta_e(\omega[N]) - \tilde{\beta}(\omega[N]) = \sum_{i=\log N+1}^{\infty} 2^{-3i}\beta_{e,i}(\omega[N]) \leq \sum_{i=\log N+1}^{\infty} 2^{-2i} = \sum_{i=1}^{\infty} 2^{-2(i+\log N)} = \frac{1}{N^2}\sum_{i=1}^{\infty} 2^{-2i} < \frac{1}{N^2}$. Lemmas 3.8 and 4.30 imply that $\tilde{\beta}_e$ is computable in $c[\mathcal{O}(e \log\log(N))) + \log\log N] = \mathcal{O}(e \log\log(N))$ space, by an algorithm querying its input in a dependency set $\{\hat{G}_{e,N}\}_{N\in\mathbb{N}}$, with $\hat{G}_{e,N} = \bigcup_{i=1}^{\log(N)} G_{e,i}$ with each $G_{e,i}$ of size $e \log\log(N)$ (the $G_{e,i}$'s come from lemma 4.30). Thus $|G_{e,N}| \leq \log(N) e \log\log(N) = \text{poly}(e \log(N))$. Therefore, $\tilde{\beta}_e$ is a $\Gamma(\text{poly}(e \log(N))$ function approximating β.

Having proved this fact, we can use the exact computation lemma 3.12 to be assured of the existence of $\bar{\beta}_e$ a $\Gamma(\text{poly}(e \log N)$ martingale such that $S^\infty[\beta_e] \subseteq S^\infty[\bar{\beta}_e]_{e\in\mathbb{N}}$.

It is now easily checked that $\{S^\infty[\bar{\beta}_e]\}_{e\in\mathbb{N}}$ is a $\Gamma(f)$ family of null sets (definition 3.16), with $f(e, N) = \text{poly}(elogN)$. We can thus finish by invoking lemma 3.17 which implies the existence of a $\Gamma(\text{poly}(f(\log N, N)) = \Gamma(\text{polylog}(N))$ martingale d such that $\bigcup_{e\in\mathbb{N}} S^\infty[\beta_e] \subseteq S^\infty[d]$. □

Corollary 4.34 *There exists d a $\Gamma(\text{polylog}\log(N))$ martingale such that if $L \in \mathcal{C}$ and $L \not\leq A_{\mathcal{C}}$, then $L \in S^\infty[d]$.*

Proof. Let d be a $\Gamma(\text{polylog}(N))$ martingale, such that $\bigcup_{e\in\mathbb{N}} S^\infty[\beta_e] \subseteq S^\infty[d]$. Such a martingale exists by lemma 4.33. Let $L \in \mathcal{C}$ be a language such that $L \not\leq A_{\mathcal{C}}$. We have to show that $L \in S^\infty[d]$. By convention 4.6, there exists $e \in \mathbb{N}$ such that $L = L_e$. Thus $L \in S^\infty[\beta_e]$ (theorem 4.28), and thus, by definition of d, $L \in S^\infty[d]$. □

We are now ready to give the main technical theorem of the paper. Intuitively it says that any complexity class satisfying a few conditions admits an almost complete problem. Fundamentally, the conditions to satisfy reduce to a lower bound on the size of complexity classes to which the theorem can be applied. In particular, the fact that the complexity classes fitting in the scheme are to satisfy at least one of two closure properties (c.f. below) forbids applying the theorem to "very" small complexity classes such as P. The next section discusses the applications and limitations of this theorem

Theorem 4.35 *Let $(\mathcal{C}, \mathcal{C}, \{\tilde{R}_e\}_{e\in\mathbb{N}})$ be a diagonalizable triplet, where $\mathcal{C}$ satisfies at least one of the following two closure properties: (1) for all space bound function s such that $\mathsf{SPACE}(s(N)) \subseteq \mathcal{C}$ it is also true that $\mathsf{SPACE}(\mathcal{O}(\max(s(N), \mathrm{polylog}(N)))) \subseteq \mathcal{C}$, (2) for all time bound function t such that $\mathtt{DTIME}(t(N)) \subseteq \mathcal{C}$ it is also true that $\mathtt{DTIME}(\mathcal{O}(\log(N)t(N) + 2^{\mathrm{polylog}(N)})) \subseteq \mathcal{C}$. Suppose that $\mu_{\mathcal{C}}$ is an $\mathcal{RBM}$ for $\mathcal{C}$ such that if β is a $\Gamma_s(\mathrm{polylog}\log(N))$ $\mathcal{BS}$, then $\mu_{\mathcal{C}}(S^\infty[\beta] \cap \mathcal{C}) = 0$. Then there is an almost complete set for $\mathcal{C}$.*

Proof.
Since $(\mathcal{C}, \mathcal{C}, \{\tilde{R}_e\}_{e\in\mathbb{N}})$ is a diagonalizable set, we can construct the language $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ of definition 4.14. Theorem 4.22 and the fact that $\mathcal{C}$ satisfies at least one of the closure properties from above imply that $A_{\mathcal{C}}$ and $B_{\mathcal{C}}$ are in $\mathcal{C}$. Since by construction $B_{\mathcal{C}} \not\leq A_{\mathcal{C}}$ it follows that $A_{\mathcal{C}}$ is not hard (and thus not complete) for $\mathcal{C}$. It remains to show that $\mu_{\mathcal{C}}(P_m(A_{\mathcal{C}}) \cap \mathcal{C}) = 1$. Theorem 4.34 implies that there exists β a $\Gamma_s(\mathrm{polylog}(N))$ $\mathcal{BS}$ such that $L \in \mathcal{C} \setminus P_m(A_{\mathcal{C}}) \Rightarrow L \in S^\infty[\beta]$. Together with the second hypothesis in the statement of the theorem, this implies that $\mu_{\mathcal{C}}(\mathcal{C} \setminus P_m(A_{\mathcal{C}})) = 0$. □

An interesting corollary is that any class for which the theorem can be applied sees its set of complete problems be small in the following sense.

Corollary 4.36 *Let $\mathcal{C}$ be a complexity class for which the theorem hereabove can be applied with an $\mathcal{RBM}$ $\mu_{\mathcal{C}}$. Let $\mathcal{CS}$ be the set of complete problems for $\mathcal{C}$, then $\mu_{\mathcal{C}}(\mathcal{CS}) = 0$.*

Proof. Let $A_{\mathcal{C}}$ be an almost complete set. Since by definition $A_{\mathcal{C}}$ is not complete, then not complete set reduces to $A_{\mathcal{C}}$ (since otherwise, by transitivity of reductions $A_{\mathcal{C}}$ would be complete). Therefore $\mathcal{CS} \cap P_m^{\leq}(A_{\mathcal{C}}) = \emptyset$, and since by definition of almost completeness $\mu_{\mathcal{C}}(P_m^{\leq}(A_{\mathcal{C}})) = 1$, it implies that $\mu_{\mathcal{C}}(\mathcal{CS}) = 0$. □

When applying theorem 4.35 to obtain almost complete problems for complexity classes such as PSPACE, we will thus have as a corollary the knowledge that not many problems in PSPACE are complete. We do not know if this result can be obtained in an easier way (i.e. without proving that there exists an almost complete set), nor if this result was known at all. Looking at the case of E (which has received much more attention from the quantitative complexity point of view), it is known that complete sets under some type of reductions are null sets (e.g. under polynomial time manyone and bounded truth table reductions), but on the other hand it is unknown whether complete problems under truth table or Turing reductions are measure null sets. (Actually, a positive answer to this question would imply a separation of the corresponding almost complete and weakly complete problems in E, thus answering an open question, c.f. [ASMRT03] for more details). Thus we suspect that for example, in the case of PSPACE, it was unknown whether the set of complete problems (under logspace manyone reductions) was large or not. Anyway, our proof of this fact is new, so it is worth mentioning, also because the generality of the statement makes it true for a wide variety of complexity classes.

5 Applications

We can now turn our attention to discussing the implications of theorem 4.35. The theorem is fairly general in the way it is stated, so it can be applied to a wide variety of complexity classes. In section 5.1, we show that it implies the existence of almost complete problems for some small complexity classes, such as PSPACE and SUBEXP. In some sense, this fulfills our expectations in terms of space bounded computations: we have reached the granularity of space efficiently computable problems. On the other hand, this is not so for time bounded computations, where we have not reached P, the class of time efficiently decidable problems. In section 5.2, we discuss the possibility of applying the theorem to other complexity classes, most interestingly, to smaller classes than PSPACE and SUBEXP, but also to big complexity classes such as E. Naturally we shall discuss the question of almost completeness in P, which is in some sense our ideal goal, at least

when regarding time bounded computations. We shall also give a look at QP, problems decidable in quasi-polynomial time. For different reasons in each case, both P and QP stay out of reach. QP is interesting, in the sense that this class, as we shall explain, gives us the feeling that our construction should be applicable to QP, but for reasons intrinsic to QP, which seem to us somehow pathological, it stays out of reach. Finally, we also point out the fact that theorem 4.35 also implies the existence of almost complete sets for bigger classes, such as EXP. This last result differs from the one from [ASMRT03] only in the sense that they prove this fact for polynomial time manyone reductions, whereas we state all our results for logspace manyone reductions.

5.1 Application to PSPACE and SUBEXP

In this section, we look at SUBEXP and PSPACE, and show how the results from the previous sections can be used to prove that these two classes admit almost complete problems. We start by making a remark on the class SUBEXP. In complexity theory, most complexity classes are defined as the inductive union of size increasing subclasses. For example, $\mathtt{P} = \bigcup_{k\in\mathbb{N}} \mathtt{DTIME}(\mathcal{O}(N^k))$. It seems that this way of defining complexity classes is somehow intrinsically related to the way we study general computational complexity. From this point of view, SUBEXP differs greatly from the rest of complexity classes, since it is not the union of size increasing subclasses, but rather the intersection of size decreasing subclasses: $\mathtt{SUBEXP} = \bigcap_{i\in\mathbb{N}} E_i$, where E_i is once again the inductive union of size increasing subclasses: $E_i = \bigcup_{j\in\mathbb{N}} \mathtt{DTIME}(2^{\mathcal{O}(N^{\frac{i-1}{ij}})})$, so that it becomes possible to study each of the E_i's with a standard computational complexity theory approach, but not SUBEXP as a whole. Therefore, we take the following usual convention, when working with SUBEXP, of saying that something is true for SUBEXP if it is true for each of the slices E_i. In particular, the statement "SUBEXP *admits an almost complete problem*" should be interpreted as meaning that "*each of the slices* E_i *of* SUBEXP *admit an almost complete problem*".

In order to make use of theorem 4.35 for PSPACE and SUBEXP, we need to see that the hypotheses required in the statement of the theorem are fulfilled. Amongst these conditions is the fact that the classes considered (PSPACE and SUBEXP) can be inserted inside of a diagonalizable triplet. We show that this holds, and start by defining standard enumerations for PSPACE and (each of the slices of) SUBEXP. The existence of these effective enumerations is standard and proved by adding to the Turing machines a gadget, called a yardstick or an alarm clock in the case of space or time bounded computations respectively.

Convention 5.1 *For any fixed* $j \in \mathbb{N}$*, we let* $\{L_i\}_{i\in\mathbb{N}}$ *be an effective enumeration of* $\mathsf{E_j}$ *and we let* $\{L'_i\}_{i\in\mathbb{N}}$ *be an effective enumeration of* PSPACE*.*

Let us also define the two following:[8]

Definition 5.2 *Let* $\mathcal{C}_{\mathsf{E_j}} = \{(1^i, 1^{|x|^{2^{\frac{i-1}{i}}}}, x) \mid M_i \text{ accepts } x \text{ in } 2^{|x|^{\frac{i-1}{ij}}} \text{ time}\}$.
Let $\mathcal{C}_{\mathtt{PSPACE}} = \{(1^i, 1^{|x|^i+i}, x) \mid M_i \text{ accepts } x \text{ in } |x|^i + i \text{ space }\}$.
Let $\tilde{R}_i(x) = (1^i, 1^{|x|^{2^{\frac{i-1}{i}}}}, x)$*. and* $\tilde{R}'_i(x) = (1^i, 1^{|x|^i+i}, x)$

Lemma 5.3 $(\mathtt{PSPACE}, \mathcal{C}_{\mathtt{PSPACE}}, \{\tilde{R}'_i\}_{i\in\mathbb{N}})$ *and* $(\mathsf{E_j}, \mathcal{C}_{\mathsf{E_j}}, \{\tilde{R}_i\}_{i\in\mathbb{N}})$ *are* diagonalizable *triplets.*

Proof. We only prove that the triplet for $\mathsf{E_j}$ is a diagonalizable triplet. The case of PSPACE is similar and easier, and left to the reader. We need to show that $\mathcal{C}_{\mathsf{E_j}}$ is complete for $\mathsf{E_j}$, and that $\{\tilde{R}_i\}_{i\in\mathbb{N}}$ is a uniform family of reductions satisfying the properties of definition 4.5. First of all we show $\mathcal{C}_{\mathsf{E_j}} \in \mathsf{E_j}$ by giving an algorithm deciding $\mathcal{C}_{\mathsf{E_j}}$ in $2^{\mathcal{O}(N^{\frac{1}{2j}})}$ time. Let $X \in \{0,1\}^*$ be the input. The algorithm needs to check that the two following points hold: verify that $X = (1^i, 1^k, x)$ for some $i \in \mathbb{N}$, with $k = |x|^{2^{\frac{i-1}{i}}}$, and check that $M(i,\cdot)$ accepts x in $2^{|x|^{\frac{i-1}{ij}}}$. Here is an algorithm accomplishing the task.

[8] We remind the reader that hereunder, M_i is the ith $\mathcal{TM}$, c.f. section 2.

1.	Compute $\lvert x\rvert^{2^{\frac{i-1}{i}}}$
2.	Check that $k = \lvert x\rvert^{2^{\frac{i-1}{i}}}$
3.	Compute $\lvert x\rvert^{\frac{i-1}{ij}}$
4.	Compute $2^{\lvert x\rvert^{\frac{i-1}{ij}}}$
5.	Set an alarm clock to ring after $2^{\lvert x\rvert^{\frac{i-1}{ij}}}$ computation steps
6.	Simulate the computation of $M(i,x)$ until the alarm clock rings
7.	Accept if $M(i,x)$ has accepted
8.	Reject if $M(i,x)$ has rejected
	or if the alarm clock rang before the simulation was finished

Let us look at the first point of the algorithm above. We need to compute the integer part of $|x|^{2^{\frac{i-1}{i}}}$. First, we compute $|x|$ (in $\log(|x|)$ time), then we compute $|x|^{i-1}$ in $\text{poly}(i\log(|x|))$ time. Then we need to extract the i-th integer root of $|x|^{2(i-1)}$. The *i-th root shifting algorithm* extracts k digits of the i-th root of an integer in $\mathcal{O}(k^3i^2)$ time. Since we want the integer part of the i-th root of $|x|^{2(i-1)}$, we have to extract $\mathcal{O}(2i\log(|x|))$ bits, which can thus be done in $\text{poly}(i\log(|x|)) = \text{poly}(|X|)$ time. The second point of the algorithm is done in $k = \mathcal{O}(|X|)$ time. Remember that j is a constant, so point 3 is computable in $\text{poly}(|X|)$ time, and since $\texttt{BIN}(2^{|x|^{\frac{i-1}{ij}}}) = 10^{|x|^{\frac{i-1}{ij}}}$, point 4 is computable in $|x|^{\frac{i-1}{ij}} < k \leq |X|$ time. Points 5 to 8 necessarily take time at most $2^{|x|^{\frac{i-1}{ij}}} = 2^{|x|^{2^{\frac{i-1}{i}}\frac{1}{2j}}} = 2^{k^{\frac{1}{2j}}} \leq 2^{|X|^{\frac{1}{2}\frac{1}{j}}}$. From this time complexity analysis, we see that $\mathcal{C}_{\mathsf{E_j}}$ qualifies as a member of $\mathsf{E_j}$. To show the completeness of $\mathcal{C}_{\mathsf{E_j}}$, it remains to see that it is hard for $\mathsf{E_j}$: let $L \in \mathsf{E_j}$. We need to show that L reduces logspace manyone to $\mathcal{C}_{\mathsf{E_j}}$. $L \in \mathsf{E_j} \Rightarrow \exists i$ s.t. L is decided by M_i in $2^{N^{\frac{i-1}{ij}}}$ time. Hence $L \leq \mathcal{C}_{\mathsf{E_j}}$ via $\tilde{R}_i$, which is easily seen to be a logspace manyone reduction. (The only difficult thing to compute $\tilde{R}_i(x) = (1^i, 1^{|x|^{2^{\frac{i-1}{i}}}}, x)$ is $1^{|x|^{2^{\frac{i-1}{i}}}}$ and more precisely (the integer part of) $|x|^{2^{\frac{i-1}{i}}}$, which is accomplished in $\mathcal{O}(\frac{i-1}{i}log(|x|)) = \mathcal{O}(\log(|x|))$ space (i is a constant!) by the *i-th root shifting algorithm*).

We next look at the family $\{\tilde{R}_i\}_{i\in\mathbb{N}}$ and verify that it satisfies the conditions required by definition 4.5. Using the previous arguments and the definition of the family we see that $L_i \leq \mathcal{C}_{\mathsf{E_j}}$ via $\tilde{R}_i$[9], the family is informally computable in $\text{poly}(i\log(n)+i)$ space, the images have a null intersection and the inverse is easy to compute. Therefore, the only things left to show are that the $\tilde{R}_i$'s are length increasing, which trivially follows from their definition, and that there is an algorithm `Check` which on input $X \in \{0,1\}^*$ and $i \in \mathbb{N}$ checks whether $X \in Im(\tilde{R}_i)$ in $\text{poly}(i\cdot\log(|X|))$ space. That is, besides trivial things, we should show that it is possible to verify that an input $X = (1^i, 1^k, x)$ satisfies $k = |x|^{2^{\frac{i-1}{i}}}$ in $\text{poly}(i\log(|X|))$ space. This can be done in $\text{poly}(i\log(|X|))$ space using the *i-th root shifting algorithm.* □

Since we have diagonalizable triplets, we can apply theorem 4.35 and obtain the following result.

Lemma 5.4 `PSPACE` *and* `SUBEXP` *have almost complete problems.*

We give only informally this easy proof.

Proof. From lemma 5.3, we know that `PSPACE` and $\mathsf{E_i}$ can be embedded inside of diagonalizable triplets. In order to apply theorem 4.35, we also need to show that `PSPACE` and $\mathsf{E_i}$ satisfy one of two closure properties (c.f. the statement of the theorem). Informally, the closure properties say that the complexity classes considered should be closed under multiplication by a logarithmic term, and that it should contain `QP` in the case of time bounded complexity classes, and for space bounded classes, it should only be that they contain `POLYLOGSPACE`, (which is a very weak condition). This is true for our two complexity classes. The final condition to satisfy is that the success sets of Γ_s(polyloglog) martingales should be null sets, for some $\mathcal{RBM}$ on the complexity

[9] In fact it holds that $\forall i \exists i_2$ s.t. $L_i = L_{i_2}$ and $L_i \leq \mathcal{C}_{\mathsf{E_j}}$ via $\tilde{R}_{i_2}$.

class considered. This last condition is satisfied as follows from lemmas 3.13 and 3.14 (and trivial space-time tradeoffs). □

From corollary 4.36, we also get the following result for free.

Corollary 5.5 *The sets of complete problems for* PSPACE *and for* SUBEXP *are null sets (in* PSPACE *and* SUBEXP *respectively).*

5.2 Other Applications and Possible Improvements

Our quest in this research has been to prove almost completeness results for small complexity classes, for which we consider PSPACE and P to be prototypes (for space and time bounded computations respectively). To some extend, we have thus succeeded in the case of space bounded computations. This is not quite so for the case of time bounded computations, where we could only prove the existence of almost complete for for SUBEXP, when having almost complete sets for P would be nicer. In this section we shall discuss why we could not improve on this, more precisely, we shall explain the difficulties encountered while trying to use our scheme to prove the existence of almost complete problems for P or, less ambitiously, for QP. Roughly speaking, in order to obtain almost complete sets for PSPACE and SUBEXP, what we did was the following. We fixed $\mathcal{C}$ to be PSPACE or (a slice of) SUBEXP . We let $\mathcal{C}_{\mathsf{E_j}} = \{(1^i, 1^M, x) \mid [M = |x|^{2^{\frac{i-1}{i}}}] \text{ and } [M_i \text{ accepts } x \text{ in } 2^{M^{\frac{1}{2}}} \text{ time }]\}$, $\mathcal{C}_{\texttt{PSPACE}} = \{(1^i, 1^M, x) \mid [M = N^i] \text{ and } [M_i \text{ accepts } x \text{ in } M \text{ space }]\}$, $\tilde{R}_i(x) = (1^i, 1^{|x|^{2^{\frac{i-1}{i}}}}, x)$ and $\tilde{R}'_i(x) = (1^i, 1^{|x|^i}, x)$ and show that $(\mathsf{E_j}, \mathcal{C}_{\mathsf{E_j}}, \{\tilde{R}_i\}_{i\in\mathbb{N}})$ and $(\texttt{PSPACE}, \mathcal{C}_{pspace}, \{\tilde{R}'_i\}_{i\in\mathbb{N}})$ are diagonalizable triplets.

What happens if we try to apply this scheme to P, the class of efficiently solvable problems? First, we construct a canonical complete language and an associated family of canonical reductions identically: $\mathcal{C}_{\texttt{P}} = \{(1^i, 1^M, x) \mid [M = |x|^i] \wedge [M_i \text{ accepts } x \text{ in time } M]\}$ and $\tilde{R}_i(x) = (1^i, 1^{|x|^i}, x)$ (which is computable in $\mathcal{O}(N^i)$ time and $\mathcal{O}(i \log N)$ space). Sadly, we cannot go any further, since lemma 4.22 does not permit to conclude that the language $A_{\mathcal{C}_{\texttt{P}}}$ (constructed using the general scheme of section 4) is in P, and thus we cannot apply theorem 4.35 to deduce the existence of an almost complete set for P, since P does not satisfy the closure property required in its statement. Furthermore, although we did not formally give the definition of an $\mathcal{RBM}$ for P, it would not be true that the success set of Γ_s(polyloglog) martingales are null sets in P as required to apply theorem 4.35. These two reasons make it look impossible to prove the existence of an almost complete set for P by adapting the proof(s) we used for PSPACE and SUBEXP: P seems really out of reach with this approach. What about QP, which is a class intermediate between P and SUBEXP? On analysis, at first sight the two obstacles encountered while looking at P seem to vanish when looking at QP: the $2^{\text{polylog}(N)}$ lower bound on the complexity of $A_{\mathcal{C}_{\texttt{QP}}}$ from theorem 4.22 (which was an obstacle in the case of P) does not disqualify $A_{\mathcal{C}_{\texttt{QP}}}$ from being in QP, and Γ_s(polyloglog) martingale have null measure success sets in QP[10] . Sadly, another difficulty arises from the fact that it seems impossible to find a canonical complete language for QP under logspace manyone reductions. Indeed, the natural canonical complete language (and its canonical reductions) for QP are the following: $\mathcal{C}_{\texttt{QP}} = \{(1^i, 1^M, x) \mid [M = 2^{\log^i(|x|)}] \wedge [M_i \text{ accepts } x \text{ in } M \text{ time }]\}$ and$\tilde{R}_i(x) = (1^i, 1^{2^{\log^i(|x|)}, x})$, which is computable in $\mathcal{O}(2^{\log^i(N)})$ time and $\mathcal{O}(\log^i(N))$ space. The reductions are not logspace manyone, but rather polylogspace manyone. This means that $(\texttt{QP}, \mathcal{C}_{\texttt{QP}}, \{\tilde{R}_i\}_{i\in\mathbb{N}})$ has no chance of being a diagonalizable triplet, but mainly because the definition was not thought of with this somehow strange case to mind. Naturally, we considered modifying this scheme, in order to adapt it to the case of QP and its canonical polylog manyone reductions. Once again, this approach fails for at least the following reason: suppose we want to compute $A_{\mathcal{C}_{\texttt{QP}}}(x)$, with $x \in I_i$. From the definition of $A_{\mathcal{C}_{\texttt{QP}}}$, this will require computing $\tilde{R}_i(x)$, but this time (in the modified scheme), $\tilde{R}_i$ is not computable in $i \log(N) + i$ space, but in $\log^i(N) + i$ space, and thus in $2^{\log^i(N)+i}$ time in the worst case, which disqualifies once again $A_{\mathcal{C}_{\texttt{QP}}}$ from being in QP (since it would require $2^{\log^i(N)+i}$ to be smaller than $2^{\log^k(N)+k}$, where i is an increasing function of N, and

[10] We give no formal justification of this fact, since anyway we fail to prove the almost completeness result for QP.

k is a constant integer). That is, even if we modify it, we can still not guarantee that $A_{\mathcal{C}}$ would be in QP for $\mathcal{C} = \mathtt{QP}$.

We claimed earlier on that we were quite satisfied with having reached, when considering space bounded computations, the granularity of PSPACE. This is true, but looking at corollary 4.34 and the proof of lemma 5.4, we see that we can in fact cover the span $P_m^{\leq}(A_{\mathcal{C}})$ of the almost complete set constructed for PSPACE with a *very* efficient martingale. In fact, this high efficiency is what permitted us to apply it to time complexity classes, through the appliance of trivial space-time tradeoffs, and to still have a reasonably enough efficient martingale to go down to SUBEXP, and to flirt with QP. This brings hope of applying theorem 4.35 to smaller space bounded complexity classes then PSPACE, probably as little as PLOGSPACE. We do not investigate this problem here, most of all because many awkward technical complications spring in mind, one of them being the fact that constructing an $\mathcal{RBM}$ for polylogspace is not trivial and would require many detailed verifications. Looking in a completely different direction, our theorem can also be applied to bigger complexity classes, such as E or EXP by choosing (e.g. in the case of E) the following diagonalizable triplet: $\mathcal{C}_{\mathtt{E}} = \{(1^i, 1^M, x) \mid [M = i|x|] \wedge [M_i \text{ accepts } x \text{ in time } 2^M]\}$ and $\tilde{R}_i(x) = (1^i, 1^{i|x|}, x)$ (which is computable in $\mathcal{O}(iN)$ time and $\mathcal{O}(\log(iN))$ space). Theorem 4.35 then yields the following corollary:

Corollary 5.6 *There exist almost complete sets for the classes* E *and* EXP *of problems decidable in exponentially linear and exponentially polynomial time.*

We should specially emphasize that this result holds for logspace manyone reductions, since [ASMRT03] prove that this holds for polynomial time manyone reductions. Also, the fact implied by corollary 4.36 that the set of complete problems for these two classes is small is already known (following a result of [RSyC95]), so we can in no way claim it as ours.

6 Conclusion

The parametrized form in which we stated our main result (theorem 4.35) makes it sensible to foresee other applications to it. In perticular we see the very efficient algorithm exhibited to compute the martingale(s) covering the complementary of the constructed almost complete set(s) (corollary 4.34) as an encouraging sign that improvement is possible in the case of space computations, (in the sense that the theorem would be applicable to smaller space bounded complexity classes then PSPACE). It is possibly sufficient to construct an $\mathcal{RBM}$ for PLOGSPACE and to verify (a possibly unpleasant bunch of) details to obtain an almost logarithmic improvement from PSPACE to PLOGSPACE by direct use of our construction.

The case of time bounded complexity classes is probably more challenging and harder to improve. Most of our algorithms were optimized in terms of space complexity. It could be that focusing on P from the start succeeded (though we see no trivial adaptation from our results) in proving the existence of almost complete problems for P. On the other hand we would not be surprised that it would be easier to adapt our results to the case of QP, which only stays out of reach for reasons which seem to come from a strangeness inherent to QP: it admits no complete problems under logspace manyone reductions.

Finally, let us point out that this line of research (the study of $\mathcal{RBM}$ related notions of completeness in small complexity classes) is rich in many more concepts to study: the existence of weakly complete problems, the size (in terms of $\mathcal{RBM}$) of the set(s) of complete problems and the different notions of completeness coming from other reductions than the ones we have strictly restrained ourselves too: logspace manyone reductions.

References

[AS94] E. Allender and M. Strauss. Measure on small complexity classes, with applications for BPP. In *Proceedings of the 35th IEEE Annual Symposium on Foundations of Computer Science*, volume 35, pages 807–818. IEEE Computer Society Press, 1994.

[AS95] E. Allender and M. Strauss. Measure on P: Robustness of the notion. In *Proceedings of the 20th Mathematical Foundations of Computer Science*, volume 969, pages 129–138. Springer, 1995.

[AS00] K. Ambos-Spies. Measure theoretic completeness notions for the exponential time classes. In *Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 152–161. Springer, 2000.

[ASMRT03] Klaus Ambos-Spies, Wolfgang Merkle, Jan Riemann, and Sebastiaan A. Terwijn. Almost complete sets. *Theoret. Comput. Sci.*, 306(1-3):177–194, 2003.

[ASMZ96] K. Ambos-Spies, E. Mayordomo, and X. Zheng. A comparison of weak completeness notions. In *Proceedings of the 11th Annual Comference on Computational Complexity*, pages 171–178. IEEE Computer Society Press, 1996.

[ASTZ97] K. Ambos-Spies, S.A. Terwijn, and X. Zheng. Resource bounded randomness and weakly complete problems. *Theoretical Computer Science*, 168:195–207, 1997.

[BDG94a] J.L Balcázar, J. Díaz, and J. Gabaró. *Structural Complexity I.* Springer-Verlag, 1994.

[BDG94b] J.L Balcázar, J. Díaz, and J. Gabaró. *Structural Complexity II.* Springer-Verlag 1990, 1994.

[Bus87] S.R. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pages 123–131, 1987.

[JL95a] D.W. Juedes and J.H. Lutz. The complexity and distribution of hard problems. *SIAM Journal on Computing*, 24(2):279–295, 1995.

[JL95b] D.W. Juedes and J.H. Lutz. Weak completeness in e and e_2. *Theoretical Computer Science*, 143:149–158, 1995.

[Jue95] D.W. Juedes. Weakly complete problems are not rare. *Computational Complexity*, 5:267–283, 1995.

[LM94] J.H. Lutz and E. Mayordomo. Measure, stochasticity, and the density of hard languages. *SIAM Journal on Computing*, 23:762–779, 1994.

[LM96] J.H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164(1–2):141–163, 1996.

[Lut92] J.H. Lutz. Almost everywhere high non-uniform complexity. *Journal of Computer and System Science*, 44:220–258, 1992.

[Lut95] J.H. Lutz. Weakly hard problems. *SIAM Journal on Computing*, 24:1170–1189, 1995.

[Lut97] J. H. Lutz. The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Retrospective II*, pages 225–260. Springer, 1997.

[May94] E. Mayordomo. *Contribution to the Study of Resource Bounded Measure.* PhD thesis, Universitat Politécnica de Catalunya, Barcelona, 1994.

[Mos02] Philippe Moser. A generalization of Lutz's measure to probabilistic classes. *Electronic Colloquium on Computational Complexity (ECCC)*, (058), 2002.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pow03] O. Powell. PSPACE contains almost complete problems. Technical Report TR03-285, Electronic Colloquium on Computational Complexity, April 2003.

[Pow04] O. Powell. A note on measuring in P. *Theoretical Computer Science*, 320(2-3):229–246, June 2004.

[RSyC95] Kenneth W. Regan, D. Sivakumar, and Jin yi Cai. Pseudorandom generators, measure theory, and natural proofs. In *Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science*, volume 36, pages 171–178. IEEE Computer Society Press, 1995.

[Str97] M. Strauss. Measure on P: Strength of the notion. *Information and Computation*, 136(1):1–23, 1997.

[Vil39] J. Ville. *Étude Critique de la Notion de Collectif*. Gauthiers-Villars, 1939.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'