Electronic Colloquium on Computational Complexity, Report No. 51 (2001)

Probabilistic abstraction for model checking: An approach based on property testing

S. Laplante * R. Lassaigne † F. Magniez * S. Peyronnet * M. de Rougemont *‡

Abstract

In model checking, program correctness on all inputs is verified by considering the transition system underlying a given program. In practice, the system can be intractably large. In property testing, a property of a single input is verified by looking at a small subset of that input. We join the strengths of both approaches by introducing to model checking the notion of probabilistic abstraction. We put forth the notion of ε-reducibility which is implicit in many property testers. Our probabilistic abstraction associates a set of small random transition systems to a program. Under some conditions, these transition systems are sufficient to guarantee that a program approximately decides on all its inputs a property like bipartiteness, k-colorability, or any first order graph properties of type $\exists\forall$. We give a concrete example of an abstraction for a program which decides bipartiteness. Finally, we show that abstraction is necessary by proving an exponential lower bound on OBDDs for approximate bipartiteness.

1 Introduction

The verification of programs is a fundamental problem in computer science, where logic, complexity and combinatorics have brought new ideas that have been influential in practical applications. We bring two general methods together: model checking, where one formally proves that a program is correct for all its inputs, up to a given length; and property testing, where a randomized algorithm makes random local checks on a particular input to decide if this input has a given property. Our approach brings the notion of approximation from property testing to model checking, and the notion of verification of the correctness of a program on all its inputs to property testing.

Formal verification is a methodology for mathematically proving properties or specifications of a program. The verification takes place once, before the program is ever executed. Given the source code, and a specification expressed in a logic-based language, the goal is to decide whether the program satisfies the specification. Model checking is an algorithmic method for deciding if a finite state program, modeled as a transition system, satisfies a specification, expressed as a formula of a temporal logic such as CTL [6]. This can be carried out in time linear in the number of states [4]. However, a program given in a classical programming language, like C, converted to a transition system, typically undergoes an exponential blowup in the size of the input. Symbolic model checking [12, 6] addresses this problem by using ordered binary decision diagrams (OBDDs, also called branching programs), which in many practical cases provide a compact representation of the transition system. Verification is decided by comparing the OBDDs. Nevertheless, in many cases, such as integer multiplication and bipartiteness, the OBDD size is exponential.

The abstraction method [5] provides a solution in some cases when the OBDD size is intractable. A large transition system is simulated by a smaller one, its abstraction, and the specification can be efficiently verified. A classical example is multiplication, where modular arithmetic is the basis of the abstraction. The goal of our paper is to extend the range of abstractions to a large family of programs on graphs.

*LRI, UMR 8623 CNRS, Université Paris–Sud, 91405 Orsay, France. Emails: {laplante,magniez,syp,mdr}@lri.fr
†Equipe de Logique, UPRESA 7056 CNRS, Université Paris 7, France. Email: lassaign@logique.jussieu.fr
‡Université Paris II, France.

ISSN 1433-8092

In the late eighties the theory of *program checking* and *self-testing/correcting*, was pioneered by the work of Blum and Kannan [2], and Blum, Luby and Rubinfeld [3]. This theory addresses the problem of program correctness by verifying carefully chosen mathematical relationships between the outputs of the program on randomly selected inputs. Rubinfeld and Sudan [13] first formulated the notion of *property testing* that arises in every such tester/corrector. One is interested in deciding whether an object has a global property ϕ by performing random local checks or queries. One is satisfied if one can distinguish with sufficient confidence between objects that satisfy ϕ and those that are ε-far from any objects that satisfy ϕ, for some confidence parameter $\varepsilon > 0$, and some distance measure. The surprising result is that when ε is fixed, this relaxation is sufficient for it to be possible to decide many properties with sublinear or even constant number of queries.

Goldreich, Goldwasser, and Ron [7, 8, 9] investigated property testing for several graph properties such as k-colorability. Alon, Fischer, Krivelevich, and Szegedy [1] showed a general result for all first order graph properties of type $\exists\forall$.

We put forth a notion that is implicit in many graph property testers: A graph property ϕ is ε-reducible to ψ if testing ψ on small random subgraphs suffices to distinguish between graphs which satisfy ϕ, and those that are ε-far from satisfying ϕ. This notion is the basis for our probabilistic abstraction, which to a program associates small random transition systems. The verification of a property on these transition systems will guarantee, under some conditions, that the program approximately satisfies its specification.

In Section 2 we review basic notions of model checking and property testing, and define ε-reducibility. In Section 3, we introduce the notion of probabilistic abstraction (Section 3.2). To be useful, an abstraction must preserve the behavior of the program. An abstraction is congruent if it preserves program correctness, and ε-robust if program correctness can be inferred from the correctness of the abstraction. The latter is an extension to abstraction of robustness introduced in [13]. We show how to derive a probabilistic abstraction using ε-reducibility (Section 3.3). We give a generic proof of ε-robustness for a large class of specifications (**Theorem 5**). Moreover, we give a sufficient condition for congruence to hold (**Theorem 6**). We establish the usefulness of our method by considering the bipartiteness problem. On the one hand, we show that a program for testing bipartiteness can be abstracted by our methods (Section 3.4). On the other hand, in Section 4 we show that abstraction is necessary, in the sense that the relaxation of the exactness of the test alone does not yield a smaller OBDD. We prove, using methods from communication complexity, that the OBDD size remains exponential for approximate bipartiteness (**Theorem 7**).

2 Preliminaries

2.1 Model checking: Transition systems and abstractions

2.1.1 Basic notions

Let P be a program with a finite set of variables $\{v_1, \ldots, v_n\}$. Each variable v_i ranges over a (finite) domain D_i. By definition, the set S of states of P is $D_1 \times \ldots \times D_n$. Its transition system is defined as follows.

Definition 1. *A transition system is a triple $M = \langle S, I, R\rangle$ where S is a set of states, $I \subseteq S$ is a set of initial states, and $R \subseteq S \times S$ is a transition relation.*

Temporal logic is used to specify properties of states, or sequences of states, of a system. A *path* of a transition system M is an infinite sequence $\sigma = (\sigma_i)_{i\in\mathbb{N}}$ of states such that $(\sigma_i, \sigma_{i+1}) \in R$, for all $i \geq 0$. Formulas of branching-time temporal logic CTL^* are defined inductively from the atomic formulas $v_i = d_i$ and built up by boolean connectives, path quantifiers and temporal operators. A detailed description of CTL^* is given in [4]. Let Θ be a CTL^* formula describing some behavior of a program. Let $M = \langle S, I, R\rangle$ be a transition system. For every $s \in S$, the notation $M, s \models \Theta$ has the usual meaning in model checking: the state s of the system M satisfies the property expressed by the formula Θ. $M \models \Theta$ means that for every

initial state $s \in I$, $M, s \models \Theta$; and for every subset $S_0 \subseteq I$, $M, S_0 \models \Theta$ means that for every initial state $s \in S_0$, $M, s \models \Theta$. When it is unambiguous, we write $M, d_i \models \Theta$ instead of $M, \{s \in I : v_i = d_i\} \models \Theta$.

The size of the OBDDs occurring in the verification of $M \models \Theta$ can be intractable. In this context, the objective of abstractions is to replace the transition system with an abstract version which is smaller and simulates the original system. For each variable, a surjection is used to reduce the size of the domain, and transitions are made between the resulting equivalence classes, as we define below.

Definition 2. *Let $M = \langle S, I, R\rangle$ be a transition system, where $S = D_1 \times \ldots \times D_n$. An* abstraction for M *is a n-tuple $h = (h_1, \ldots, h_n)$, where $h_i : D_i \to \widehat{D}_i$ is a surjection, and $\widehat{D}_i$ is any set. The* minimal transition system of M with respect to h *is the transition system $\widehat{M}_{\min} = \langle \widehat{S}, \widehat{I}, \widehat{R}_{\min}\rangle$ such that $\widehat{S} = \widehat{D}_1 \times \ldots \times \widehat{D}_n$, $\widehat{I}_{\min} = h(I)$, and*

$$(\widehat{s}, \widehat{s}') \in \widehat{R}_{\min} \quad \Longleftrightarrow \quad \exists (s, s') \in S^2,\ h(s) = \widehat{s} \wedge h(s') = \widehat{s}' \wedge (s, s') \in R.$$

2.1.2 Producing abstractions

In general, it is very difficult to construct $\widehat{M}_{\min}$ because the full description of the transition system M is needed in order to carry out the abstraction. Nevertheless, one can produce an approximation directly from the representation of the program.

Definition 3. *Let $M = \langle S, I, R\rangle$ be a transition system, and let $h : S \to \widehat{S}$ be an abstraction for M. A transition system $\widehat{M} = \langle \widehat{S}, \widehat{I}, \widehat{R}\rangle$* approximates *$M$ with respect to h ($M \sqsubseteq_h \widehat{M}$ for short) if and only if $\widehat{I} = h(I)$, and $\widehat{R}_{\min} \subseteq \widehat{R}$.*

For any formula $\phi(v_1, \ldots, v_n)$, we define the operator $[\cdot]$, verifying $[R] = \widehat{R}_{\min}$:

$$[\phi](\widehat{v}_1, \ldots, \widehat{v}_n) \stackrel{\text{def}}{=} \exists v_1, \ldots, \exists v_n,\ \big(\textstyle\bigwedge_{i=1}^{n} \widehat{v}_i = h(v_i)\big) \wedge \phi(v_1, \ldots, v_n).$$

To obtain an approximation, we apply $[\cdot]$ at the atomic level, *i.e.* to the primitive relations P of the program and their negations $\neg P$. We obtain the approximation operator $\mathcal{A}$ by induction on first order formulas that we assume to be in negative normal form (negations pushed down to the atomic level).

Theorem 1 ([5]). *Let $M = \langle S, I, R\rangle$ be a transition system, and let $h : S \to \widehat{S}$ be an abstraction for M. Then $\mathcal{A}(M) = \langle \widehat{S}, \mathcal{A}(I), \mathcal{A}(R)\rangle$ approximates M.*

2.1.3 Relations between abstract model and concrete model

Let $\widehat{M}$ be an approximation of M. Suppose that $\widehat{M} \models \Theta$. What can we conclude on the concrete model M? To answer, let us first consider the following $\mathcal{C}$ and $\mathcal{D}$ correspondences between formulas. These correspondences preserve boolean connectives, path quantifiers, and temporal operators, and transform the atomic formulas as follows:

$$\mathcal{C}(\widehat{v}_i = \widehat{d}_i) \stackrel{\text{def}}{=} \bigvee_{d_i : h_i(d_i) = \widehat{d}_i} (v = d_i) \qquad\qquad \mathcal{D}(v_i = d_i) \stackrel{\text{def}}{=} (\widehat{v}_i = h_i(d_i))$$

We denote by $\forall\mathrm{CTL}^*$ and $\exists\mathrm{CTL}^*$ the universal fragment and the existential fragment of CTL^*.

Theorem 2 ([5]). *Let $M = \langle S, I, R\rangle$ be a transition system. Let $h : S \to \widehat{S}$ be an abstraction for M, and let $\widehat{M}$ be such that $M \sqsubseteq_h \widehat{M}$. Let Θ be a $\forall\mathrm{CTL}^*$ formula on $\widehat{M}$, and Θ' be a $\exists\mathrm{CTL}^*$ formula on M. Then*

$$\widehat{M} \models \Theta \quad \Longrightarrow \quad M \models \mathcal{C}(\Theta), \qquad \text{and} \qquad M \models \Theta' \quad \Longrightarrow \quad \widehat{M} \models \mathcal{D}(\Theta').$$

This theorem is a consequence of the following lemma which is more convenient for practical use. For every path $\sigma = s_0 s_1 s_2 \ldots$ in M, the path $h(s_0)h(s_1)h(s_2)\ldots$ will be denoted by $h(\sigma)$.

Lemma 1 ([5]). *Let M be a transition system. Let $h : S \to \widehat{S}$ be an abstraction for M, and let $\widehat{M}$ be such that $M \sqsubseteq_h \widehat{M}$. If σ is a path in M, then $h(\sigma)$ is a path in $\widehat{M}$.*

2.2 Property testing

We consider only undirected, simple graphs (no multiple edges or self-loops). For a graph G, we denote by V_G its vertex set, by E_G its edge set, and by n the cardinality $|V_G|$ of V_G. When there is no ambiguity, we will simply write V and E instead of V_G and E_G. We will be interested in properties which are invariant under graph isomorphisms, so labeling is not important. In the remainder of the paper we will use the following distance measure: for any two graphs G and G' on the same n-vertex set, $\mathsf{Dist}(G, G')$ is the fraction of the n^2 edges on which the graphs disagree. Our main results hold for any distance measure.

Let ϕ be a graph property. Then $G \models \phi$ means that G has the property ϕ. In general, an *ε-test* for ϕ is a probabilistic algorithm that accepts every object with property ϕ, and rejects with probability $2/3$ every object that has distance more than ε from any object having the property.[1] The property ϕ is called *testable* if for every $\varepsilon > 0$, there exists an ε-test for ϕ whose total number of queries depends on ε, but does not depend on the size of the graph. In several cases, the proof of testability is based on a mathematical reduction between two properties. The notion of ε-reducibility highlights this idea. This notion is central to the design of our abstractions. Denote by G_π the vertex-induced subgraph of G on the vertex set $\pi \subseteq V_G$. For every property ϕ on n-vertex graphs, we denote by ϕ_ε the property that for every n-vertex graph G,

$$G \models \phi_\varepsilon \quad \Longleftrightarrow \quad \exists H,\ \mathsf{Dist}(G, H) \leq \varepsilon \text{ and } H \models \phi.$$

Definition 4. *Let $\varepsilon > 0$, $1 \leq k \leq n$ be two integers, and ϕ (resp. ψ) be a property on n-vertex graphs (resp. k-vertex graphs). For every graph G, let Π denote the set of all $\pi \subseteq V_G$ such that $|\pi| = k$. Then ϕ is* ε-reducible to ψ *if and only if for every n-vertex graph G,*

$$\begin{aligned} G \models \phi \quad &\Longrightarrow \quad \forall \pi \in \Pi,\ G_\pi \models \psi, \\ G \models \neg\phi_\varepsilon \quad &\Longrightarrow \quad \mathop{\mathbf{Pr}}_{\pi \in \Pi}[G_\pi \models \psi] \leq 1/3. \end{aligned}$$

For example, we can recast the results of k-colorability and bipartiteness [7] in terms of ε-reducibility.

Theorem 3 ([7]). *For all $k \geq 3$, $\varepsilon > 0$,*
(1) k-colorability is ε-reducible to k-colorability on $O(k^4 \log^2(k/\varepsilon)/\varepsilon^6)$-vertex graphs.
(2) bipartiteness is ε-reducible to bipartiteness on $O(\log^2(1/\varepsilon)/\varepsilon^3)$-vertex graphs.

Recently, Alon, Fischer, Krivelevich, and Szegedy [1] showed that all first order graph properties of type $\exists\forall$ have an ε-tester. Their results can also be recast in terms of ε-reducibility, as follows. Note, however, that in this result, the function f is a tower of towers.

Theorem 4 ([1]). *There exists a function $f : \mathbb{R}_+ \to \mathbb{R}_+$, such that every first order graph property ϕ of type $\exists\forall$ is ε-reducible to some property on $O(f(t + 1/\varepsilon))$-vertex graphs, where ϕ has t internal variables.*

3 Verification of graph properties

3.1 Context

Consider the following special case of a formula we wish to verify. Let P be a program that computes some boolean function. We will use two special boolean variables `ack` (acknowledge) and `ret` (return). During the computation, the boolean variable `ack` is *false*; at the end of the execution `ack` is *true*. The output of the program is the boolean variable `ret`. Then if ϕ is some property on graphs and M is the transition system of the program, we would like to check the following,

$$\forall G, \quad M, G \models \exists \left((\neg \mathtt{ack})\, \mathbf{U} (\mathtt{ack} \wedge \mathtt{ret})\right) \quad \Longrightarrow \quad G \models \phi.$$

[1] We may also consider two-sided error, and the choice of the success probability $2/3$ is of course arbitrary.

On the RHS of the implication, G represents a structure in which ϕ is interpreted, and on the LHS, G is the value of an input variable of the program. Here, the symbol $\exists$ is the existential path quantifier, and $\mathbf{U}$ is the temporal operator *until*. Since the program is deterministic, there is only one path when G is fixed.

More generally, our framework applies to the following type of formulas:

$$\forall G, \quad M, G \models \Theta \quad \Longrightarrow \quad G \models \phi, \tag{1}$$

where the input may include the graph G together with auxiliary data, Θ is a CTL^* formula, and ϕ is a graph property. Here, the existential (or universal) quantifier applies to all execution paths generated by different initial states (typically, different initial values of variables, including auxiliary data), and possibly by nondeterministic transitions.

For the graph properties that we consider, such as bipartiteness, the OBDDs have exponential size. As we show in Section 4, the ε-relaxation of property testing is not sufficient to reduce the size of the OBDD. We use ε-reducibility to construct probabilistic abstractions, yielding smaller, even constant-size OBDDs.

3.2 Probabilistic abstraction

Definition 5. *Let M be a transition system. A* probabilistic abstraction of M *is a triple $(\mathcal{H}, \mathcal{M}, \mu)$, where $\mathcal{H}$ is a set of abstractions for M, $\mathcal{M}$ is a functional which maps every $h \in \mathcal{H}$ into a transition system $\widehat{M}^h = \mathcal{M}(h)$ such that $M \sqsubseteq_h \widehat{M}^h$, and μ is a probability distribution over $\mathcal{H}$*

Let Θ be a CTL^* formula on M, and ψ be a graph property. Then any probabilistic abstraction of M induces the following probabilistic test, where $\widehat{G}^h$ denotes a graph, and the operator $\mathcal{D}$ is applied with respect to the appropriate abstraction.

Generic Test $\big((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi\big)$

1. Choose an element $h \in \mathcal{H}$ according to μ.
2. Accept if (and only if)
$$\forall \widehat{G}^h, \quad \widehat{M}^h, \widehat{G}^h \models \mathcal{D}(\Theta) \quad \Longrightarrow \quad \widehat{G}^h \models \psi.$$

The probability that the test rejects will be denoted by $\mathsf{Rej}\big((\mathcal{H}, \mathcal{M}), \mu, \Theta, \psi\big)$. The distribution μ will be omitted when it denotes the uniform probability distribution. To be useful, a probabilistic abstraction should be both ε-robust and congruent, in which case we say that it is an ε-abstraction.

Definition 6. *Let M be a transition system, $\varepsilon > 0$, Θ be a CTL^* formula, and let ϕ, ψ be two graph properties. A probabilistic abstraction $(\mathcal{H}, \mathcal{M}, \mu)$ of M is* ε-robust with respect to (Θ, ϕ, ψ) *if*

$$(\exists G, \quad M, G \models \Theta \quad \text{and} \quad G \models \neg\phi_\varepsilon) \quad \Longrightarrow \quad \mathsf{Rej}\big((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi\big) \geq \frac{2}{3}.$$

Definition 7. *Let M be a transition system, Θ be a CTL^* formula, and let ϕ, ψ be two graph properties. A probabilistic abstraction $(\mathcal{H}, \mathcal{M}, \mu)$ of M is* congruent with respect to (Θ, ϕ, ψ) *if*

$$(\forall G, \quad M, G \models \Theta \quad \Longrightarrow \quad G \models \phi) \quad \Longrightarrow \quad \mathsf{Rej}\big((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi\big) = 0.$$

Definition 8. *Let M be a transition system, Θ be a CTL^* formula, and let ϕ, ψ be two graph properties. A probabilistic abstraction $(\mathcal{H}, \mathcal{M}, \mu)$ of M is an* ε-abstraction for (Θ, ϕ, ψ) *if it is both ε-robust and congruent with respect to (Θ, ϕ, ψ).*

3.3 Constructing ε-abstractions

We now explain how to construct ε-abstractions based on ε-reducibility. Fix $\varepsilon > 0$, $n \geq 1$, and assume that ϕ on n-vertex graphs is ε-reducible to ψ on k-vertex graphs, for some $1 \leq k \leq n$. We give a generic proof of robustness of our probabilistic abstraction, and we isolate a sufficient condition which implies congruence. Under this condition, we obtain an ε-abstraction.

Since we relax ϕ with respect to ε, we can decompose our initial specification (1) into the following family of reduced specifications:

$$\{(\forall G, \quad M, G \models \Theta \quad \Longrightarrow \quad G_\pi \models \psi) : \pi \in \Pi\}.$$

For every π, the corresponding reduced specification can now be subject to an abstraction h_π. We require that the abstraction of G be exactly G_π, that is, $\widehat{G}^\pi = G_\pi$. Let $\widehat{M}^\pi$ be such that $M \sqsubseteq_{h_\pi} \widehat{M}^\pi$.

We define the (uniform) probabilistic abstraction $(\mathcal{H}, \mathcal{M})$ (also denoted by $(\Pi, \mathcal{M})$) as $\mathcal{H} = \{h_\pi : \pi \in \Pi\}$ and $\mathcal{M}(h_\pi) = \widehat{M}^\pi$, for every $\pi \in \Pi$. This leads to the following test:

Graph Test$\big((\Pi, \mathcal{M}), \Theta, \psi\big)$
1. Randomly choose a subset of vertices $\pi \in \Pi$.
2. Accept if (and only if)
$$\forall \widehat{G}^\pi, \quad \widehat{M}^\pi, \widehat{G}^\pi \models \mathcal{D}(\Theta) \quad \Longrightarrow \quad \widehat{G}^\pi \models \psi.$$

We show that if Θ is an $\exists\mathrm{CTL}^*$ formula, then our probabilistic abstraction is ε-robust.

Theorem 5. *Let Θ be a $\exists\mathrm{CTL}^*$ formula. Let $\varepsilon > 0$ be a real, and let ϕ on n-vertex graphs be ε-reducible to ψ on k-vertex graphs. Let $(\Pi, \mathcal{M})$ be a probabilistic abstraction such that $\widehat{G}^\pi = G_\pi$, for every $\pi \in \Pi$. Then $(\pi, \mathcal{M})$ is ε-robust with respect to (Θ, ϕ, ψ).*

Proof. Let G be such that $M, G \models \Theta$ and $G \models \neg\phi_\varepsilon$. By Theorem 2, $\widehat{M}^\pi, \widehat{G}^\pi \models \mathcal{D}(\Theta)$, for every $\pi \in \Pi$. Moreover, by definition of ε-reducibility we know that $\mathbf{Pr}_{\pi\in\Pi}\left[\widehat{G}^\pi \models \psi\right] \leq 1/3$. Therefore

$$\mathbf{Pr}_{\pi\in\Pi}\left[\widehat{M}^\pi, \widehat{G}^\pi \models \mathcal{D}(\Theta) \quad \Longrightarrow \quad \widehat{G}^\pi \models \psi\right] \leq \tfrac{1}{3}.$$

We conclude by observing that $1 - \mathsf{Rej}(M)$ is upper bounded by the term on the LHS of the inequality. □

Having shown that the abstraction is ε-robust, we establish a sufficient condition for congruence.

Definition 9. *Let h be an abstraction, and let $\widehat{M}$ be such that $M \sqsubseteq_h \widehat{M}$. Then the approximation $\widehat{M}$ is* exact with respect to Θ *if and only if for every graph $\widehat{G}$,*

$$\widehat{M}, \widehat{G} \models \mathcal{D}(\Theta) \quad \Longrightarrow \quad \exists H, \quad \widehat{H} = \widehat{G} \quad \text{and} \quad M, H \models \Theta.$$

Theorem 6. *Let Θ be a $\exists\mathrm{CTL}^*$ formula. Let $\varepsilon > 0$ be a real, and let ϕ on n-vertex graphs be ε-reducible to ψ on k-vertex graphs. Let $(\Pi, \mathcal{M})$ be a probabilistic abstraction such that $\widehat{G}^\pi = G_\pi$ and $\widehat{M}^\pi$ is an exact approximation with respect to Θ, for every $\pi \in \Pi$. Then $(\pi, \mathcal{M})$ is congruent with respect to (Θ, ϕ, ψ).*

Proof. Fix $\pi \in \Pi$. Let $\widehat{G}^\pi$ be a k-vertex graph such that $\widehat{M}^\pi, \widehat{G}^\pi \models \mathcal{D}(\Theta)$. From the exactness of $\widehat{M}^\pi$, there exists a graph H such that $\widehat{H}^\pi = \widehat{G}^\pi$ and $M, H \models \Theta$. Therefore, from the hypotheses we get $H \models \phi$. The ε-reducibility of ϕ to ψ implies that $H_\pi \models \psi$, that is, $\widehat{G}^\pi \models \psi$. This shows that for all $\pi \in \Pi$ and $\widehat{G}^\pi$,

$$\widehat{M}^\pi, \widehat{G}^\pi \models \mathcal{D}(\Theta) \quad \Longrightarrow \quad \widehat{G}^\pi \models \psi.$$

□

3.4 An ε-abstraction for bipartiteness

In this section, we give a short program for bipartiteness, and an ε-abstraction for this program. Our program is written in a simple language that manipulates bit, bounded integer and finite array variables, using basic instructions: while statements, conditionals, assignments. Recall that we have an implicit variable `ack` which is set to `false` at the beginning of the program and is set to `true` at the end of the computation. There is an additional instruction `RETURN` and a variable `ret`. `RETURN b` sets `ack` to *true* and `ret` to `b`. To verify properties on the behavior of a program, we must know values of the variables at certain steps of the computation, called *control points*. The control points are on lines labeled by integers. During the computation, one can observe the states of the program at the beginning of labeled lines. When the program terminates, the transition system loops with an infinite sequence of transitions on the last state.

The transition between two control points is translated into a boolean formula. This formula defines a relation between new and old values of variables of the program. The disjunction of all these formulas defines the global transition relation of the program. The definition of the global transition relation is illustrated on the following toy example.

```
INPUT b : BOOLEAN
1 : b = not b
2 : RETURN b
```

A state of the system is a tuple $(\mathtt{PC}, \mathtt{b}, \mathtt{ack}, \mathtt{ret})$, where `PC` is an integer which denotes the current line. The formula which defines the initial states is $(\mathtt{ack} = \mathit{false}) \wedge (\mathtt{PC} = 1)$. The notation $\mathbf{i} \mapsto \mathbf{j}$ stands for $(\mathtt{PC} = i) \wedge (\mathtt{PC'} = j)$. The two formulas expressing the transition relation are:

$$\mathbf{1} \mapsto \mathbf{2}\text{:}\ (\mathtt{b'} = \neg \mathtt{b}) \qquad\qquad \mathbf{2} \mapsto \mathbf{2}\text{:}\ (\mathtt{ack'} = \mathit{true}) \wedge (\mathtt{ret'} = \mathtt{b})$$

We consider a function that decides, given a graph `G` and a coloring `Color`, if `Color` is a bipartition for `G`. The graph `G` is represented by the upper triangular entries of a matrix and `Color` by an array.

```
FUNCTION CHECK-PARTITION

  CONSTANT INTEGER n=10000

  INPUT G : ARRAY[n,n] of BOOLEAN
  INPUT Color : ARRAY[n] of BOOLEAN

  VAR u,v : INTEGER 1..n+1
```

```
1: u=2
2: WHILE u<=n DO
     v=1
3:   WHILE v<=u-1 DO
4:      IF G[u,v] && (Color[u]=Color[v])
5:         RETURN false
        v=v+1
     u=u+1
6: RETURN true
```

We want to verify that, for every input `G`, if there exists an input `Color` for which the program accepts, then `G` represents a bipartite graph. More formally, we want to verify the following property:

$$\forall G, \quad M, G \models \exists\big((\neg \mathtt{ack})\, \mathbf{U} (\mathtt{ack} \wedge \mathtt{ret})\big) \quad \Longrightarrow \quad G \text{ is bipartite.} \tag{2}$$

Here, $\exists$ ranges over all the possible initial values of `Color`. For each π we define the abstraction which maps `G` to the subgraph $\mathtt{G}_\pi$, `Color` to the coloring on the subset of vertices induced by π, and $\mathtt{u}, \mathtt{v}$ refer to the vertices of the subgraph as follows:

$$\mathtt{u}_\pi = \begin{cases} \mathtt{u}, & \text{if } \mathtt{u} \in \pi, \\ \min\{w : \forall t, w \le t \le \mathtt{u} \implies t \notin \pi\}, & \text{otherwise.} \end{cases}$$

For our abstraction, the following lemma holds (proof in Appendix A).

Lemma 2. *For every graph G and $\pi \in \Pi$, $\mathcal{A}^\pi(M, G)$ is exact with respect to the $\exists\mathrm{CTL}^*$ formula in* (2).

Since the size of π is fixed, our abstraction induces a constant size OBDD. By Lemma 2, Theorems 6, 3, and 5, we know that our probabilistic abstraction is an ε-abstraction, so **Graph Test** can be used for checking the validity of (2). This test uses constant size OBDDs, and it satisfies:

(1) If the program satisfies the specification (2), **Graph Test** always accepts.
(2) If the program does not verify the ε-relaxation of (2), **Graph Test** rejects with probability $\geq \frac{2}{3}$.

4 Lower bound for approximate bipartiteness

In this section, we show how the communication complexity lower bound of [10] can be modified to yield a lower bound on OBDD size for the relaxed version of bipartiteness. This establishes that in the case of bipartiteness, reducing the size of the OBDD cannot be achieved solely by relaxing the exactness of the result. A lower bound for the relaxed version of connectivity can also be obtained using similar ideas.

4.1 Preliminaries

A graph $G = (V, E)$ is *k-bipartite* if there is a set of edges $e_1, \ldots, e_{k'}$ with $k' \leq k$ such that $G' = (V, E \setminus \{e_1, \ldots, e_{k'}\})$ is bipartite. In particular, graph is 0-bipartite if and only if it is bipartite.

The *k-bipartiteness promise problem* is a partial function F over graphs G with

$$F(G) = \begin{cases} 1 & \text{if } G \text{ is bipartite} \\ 0 & \text{if } G \text{ is not } k\text{-bipartite} \\ \perp & \text{otherwise.} \end{cases}$$

In the following four sections, we will show a lower bound on the OBDD size for the promise problem associated with k-bipartiteness. An OBDD solves the k-bipartiteness promise problem if its output agrees with F whenever F is defined, and it outputs either 0 or 1 whenever f is undefined on G (i.e., $F(G) = \perp$).

Theorem 7. *Any OBDD solving the k-bipartiteness promise problem has width at least $2^{\Omega((n-2\sqrt{k+1})\log(n-2\sqrt{k+1}))}$. When $k = \varepsilon n^2$, the width is $2^{\Omega((1-2\sqrt{\varepsilon})n\log((1-2\sqrt{\varepsilon})n))}$.*

We denote by $[m]$ the set $\{1, \ldots, m\}$. $S \dot\cup T$ denotes disjoint union of sets S and T. A *partition* of a finite set S is a set of non-empty parts $S_1, S_2, ...$ whose disjoint union $S_1 \dot\cup S_2 \dot\cup \cdots$ equals S. The number of partitions of a set S containing n elements is B_n, the nth Bell number, where B_n is $2^{\Omega(n \log n)}$.

There will be two kinds of partitions in the proof: partitions of a subset of vertices in the HMT graphs (defined below), and partitions of the edge variables of the graph (also explained below). To avoid confusion we call the latter a coloring instead of a partition, and use the letters R (red variables given to Player I) and Y (yellow variables given to Player II) to denote the color sets. In the remainder, we only consider colorings which divide the edge variables into two sets of equal size.

Let $f : \{0,1\}^N \to \{0,1\}$ be a boolean function which two players wish to compute. Let $R \dot\cup Y$ be a coloring of the N input variables. Player I's input x corresponds to the variables of R, Player II's input y corresponds to the variables of Y. In a *one-way communication protocol*, Player I sends one message to Player II, who outputs the value of $f(x, y)$, where it is understood that the variables are reordered appropriately according to R, Y. The *communication $\kappa_{\to}^{R:Y}(P; x, y)$ incurred by a one way-communication protocol* P on input x, y for the coloring R, Y is the number of bits sent by Player I.

For a fixed input length N, the *one-way communication complexity of f* for coloring R, Y is $\kappa_{\to}^{R:Y}(f) = \min_P \max_{x,y}\{\kappa_{\to}^{R:Y}(P; x, y)\}$. The *one-way communication complexity for the best-case coloring of variables* is $\kappa_{\to}^{best}(f) = \min_{R \dot\cup Y}\{\kappa_{\to}^{R:Y}(P)\}$.

Proposition 1 ([11, Page 144]).

(1) If f has an OBDD of width at most w, then $\kappa_{\to}^{best}(f) \leq \log w$.

(2) Let M_f be the communication matrix associated with the boolean function f whose variables are colored by R, Y. Then $\kappa_{\to}^{R:Y}(f) \geq log(l)$, where l is the number of distinct lines in M_f.

4.2 Sketch of the proof

Hajnal, Maass and Tur´an [10] show that the communication complexity of connectivity and bipartiteness is $\theta(n \log n)$, for arbitrary colorings of the variables into two sets of equal size. To get the lower bound on OBDDs, we show that the one-way communication complexity for arbitrary colorings is large.

We give a high-level sketch the proof, which follows the same structure as that of [10]. Let G be a graph with n vertices. The variables in the communication problem are pairs of vertices: the variable is set to 1 whenever the corresponding edge is in the graph, and to 0 otherwise. Each player is given half of the variables. The goal of the communication protocol is to determine whether the graph formed by the union of edges from the players' variables forms a bipartite graph.

In [10], given any coloring of the $(n^2-n)/2$ edges, a large family of HMT graphs is embedded within the coloring. **(Definition 10)**. They show **(Lemma 3)** how to obtain a family of HMT graphs, parameterized by two partitions of the set B, in such a way that and each player "owns" one of the partitions (that is, the edges in each player's set of variables encodes a partition.) Each graph in this family is bipartite if and only if a certain property holds for the two underlying partitions. The lower bound for bipartiteness is obtained by giving a lower bound on the underlying property of partitions.

We modify the HMT graphs to accommodate the relaxed version of bipartiteness. We show **(Lemma 4)** that the relaxed version of bipartiteness corresponds to a property on partitions. Finally, **(Lemma 5)** we give a lower bound on the number of distinct rows of the communication matrix of the relaxed bipartiteness problem, yielding a lower bound on the one-way communication complexity, hence on OBDD size.

4.3 HMT graphs

In the communication protocol, the variables are shared evenly between two players according to a coloring $R\dot{\cup}Y$. This coloring can be seen as a coloring of the complete graph on $N = (n^2 - n)/2$ vertices.

The construction of [10] produces a large family of graphs (parameterized by P, P') that can be embedded into the coloring, in such a way that Player I's (red) edges represent a partition P of a set of vertices of size $\Omega(n)$, and Player II's (yellow) edges represent a partition of P' of the same set.

Definition 10 ([10]). *An* HMT graph family *for a coloring R, Y of the $(n^2 - n)/2$ variables is a family of graphs $\{G_{P,P'}\}$ over n vertices, with $A\dot{\cup}B\dot{\cup}C \subseteq [n]$ and $|B| = \Omega(n)$, and such that for any pair of partitions P, P' of the vertices in B, the graph $G_{P,P'}$ has the following properties.*

(1) B is an independent set of $G_{P,P'}$, there are no edges between A and C, and the vertices $[n] \setminus A\dot{\cup}B\dot{\cup}C$ are isolated vertices;[2]
(2) All edges between A and B (resp., B and C) are red (resp. yellow).
(3) For every pair x, y included in some part of P (resp., P'), there is a red (resp. yellow) path of length 4 between x and y in $G_{P,P'}|_{A\cup B}$ (resp. $G_{P,P'}|_{B\cup C}$)

Lemma 3 ([10]). *Fix any complete graph on n vertices, with half the edges colored red and half yellow. Then if n is large enough, there is an HMT graph family for this coloring of the edge variables.*

A proof sketch is given in Appendix B.

4.4 Bipartiteness and partitions

For a fixed coloring R, Y of the edge variables, we show how to construct instances of k-bipartiteness from a family of HMT graphs for this coloring. For any two partitions P, P' of a set X, $P\vee P'$ is the finest partition which refines both P and P'. A key observation is that the connected components of an HMT graph $G_{P,P'}$ correspond exactly to the parts of $P\vee P'$. The following holds for any HMT graph $G_{P,P'}$.

Proposition 2 ([10]). *Let $B = \{v_1, v_2, \ldots\}$ be the subset of vertices of $G_{P,P'}$ from Definition 10. Let $G^0_{P,P'}$ be $G_{P,P'}$ to which the single edge $\{v_1, v_2\}$ has been added. Then $G^0_{P,P'}$ is bipartite if and only if $\{v_1, v_2\}$ is not included in any part $Q \in P\vee P'$.*

[2] For the proof of k-connectivity, the isolated vertices must be connected to the rest of the graph. We do not need this for the lower bound on bipartiteness.

This is because the addition of an edge within a part $Q \in P \vee P'$ creates an odd cycle in the graph. No cycle is created if the edge "crosses over" two parts of $P \vee P'$ because the parts of $P \vee P'$ correspond to connected components of $G_{P,P'}$.

For the k-bipartiteness problem, instead of adding a single, fixed, edge $\{v_1, v_2\}$ to each graph $G_{P,P'}$, we add a fixed bipartite graph with $k+1$ edges within the first $2\sqrt{k+1}$ vertices $v_1, v_2, \ldots$ of B, in such a way that edges added in this way only go between even and odd vertices in this set of vertices. These edges are labeled $e_0, \ldots, e_k$, and are the same for all the graphs $G_{P,P'}$ in the family. We call the resulting graph (i.e., $G_{P,P'}$ together with the $k+1$ additional edges) $G^k_{P,P'}$.

To state the relationship between partitions and k-bipartiteness of $G_{P,P'}$, we define an auxiliary graph $H^k_{P,P'}$. The vertices of the graph are the parts of $P \vee P'$. There is an edge in $H^k_{P,P'}$ between two parts $Q, Q' \in P \vee P'$ whenever some edge e_i crosses over Q, Q', that is, $e_i = \{a, b\}$, with $a \in Q$ and $b \in Q'$. H_k can have multiple edges. There are two ways that an odd cycle can be created when adding the $k+1$ new edges: either by adding an edge within a part of $P \vee P'$, or by creating an odd cycle in $H^k_{P,P'}$.

Lemma 4. *Let $B = \{v_1, v_2, \ldots\}$ be the subset of vertices of $G_{P,P'}$ from Definition 10 and let $G^k_{P,P'}, H^k_{P,P'}$ be defined as above.*

(1) $G^k_{P,P'}$ is bipartite iff $H^k_{P,P'}$ has $k+1$ edges (counting multiple edges) and has no even cycle.

(2) $G^k_{P,P'}$ is not k-bipartite iff for all edges $e \in \{e_0, \ldots e_k\}$ there is some part $Q_e \in P \vee P'$ such that $e \subseteq Q_e$ (or equivalently, $H^k_{P,P'}$ has no edges).

Proof. Part 1 follows from the fact that the only way to create an odd cycle in the graph is to add an edge within a part of $P \vee P'$, or to create an odd cycle in $H^k_{P,P'}$. The new edges form a bipartite subgraph so they cannot form an odd cycle on their own.

For Part 2, notice that the odd cycles arising from an odd cycle in $H^k_{P,P'}$ can be removed by removing a single edge. Therefore, if this case occurs, the resulting graph is k'-bipartite for some $k' < k$. On the other hand, if $H^k_{P,P'}$ is empty, then $G^k_{P,P'}$ contains $k+1$ disjoint odd cycles: this is because the paths in Lemma 6 are pairwise disjoint. Therefore, $k+1$ edges must be removed in order for the graph to be bipartite. □

4.5 Distinct lines in the communication matrix

In this section, we consider the communication matrices underlying the k-bipartiteness problem. By Proposition 1, a lower bound on the number of distinct lines in the communication matrix results in a lower bound on OBDD width (and *a fortiori* on OBDD size.)

Since we are considering a promise version of k-bipartiteness, the communication matrices we consider will have entries 0 and 1 when the promises are met, and $\star$ when the computation can output either 0 or 1. Therefore, when we prove that there is a large number of distinct lines, we only consider two lines to be distinct if on some column, one line contains a 0 and the other contains a 1.

We derive a communication matrix from Lemma 4. By Lemma 3, this matrix is a submatrix of the communication matrix for the promise version of k-bipartiteness. More precisely, we define a matrix M whose rows are labeled by a partition P and columns by a partition P'. The number of rows and columns is B_s, the sth Bell number, where s is $\Omega(n)$.

$$M_{P,P'} = \begin{cases} 1 & \text{if } H^k_{P,P'} \text{ contains no edges} \\ 0 & \text{if } H^k_{P,P'} \text{ contains } k+1 \text{ edges and contains no even cycle} \\ \star & \text{otherwise.} \end{cases}$$

Lemma 5. *The matrix M has $2^{\Omega((n-2\sqrt{k+1}) \log (n-2\sqrt{k+1}))}$ distinct lines.*

The proof is in Appendix B.

References

[1] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.

[2] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

[3] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comp. and Syst. Sci.*, 47(3):549–595, 1993.

[4] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[5] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[7] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[8] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proc. 27th STOC*, pages 406–415, 1997.

[9] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. In *Proc. 29th STOC*, pages 289–298, 1998.

[10] A. Hajnal, W. Maass, and G. Tur´an. On the communication complexity of graph properties. In *Proc. of the 20th STOC*, pages 186–191, 1988.

[11] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[12] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[13] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comp.*, 25(2):23–32, April 1996.

[14] E. Szemer´edi. Regular partitions of graphs. InÉditions du CNRS, *Problèmes combinatoires et théorie des graphes*, pages 399–401, 1978.

A Proof of Lemma 2

Proof of Lemma 2. The proof is in two parts. First, we will prove that the abstraction which maps $\mathtt{G} \mapsto \mathtt{G}_\pi$, and preserves the other variables, is valid for every π. This is the most difficult part. Then we show how it can be extended to the complete abstraction.

The boolean formulas that describe the transition system associated to the program are:

Initial states: $(\mathtt{ack} = \mathit{false}) \wedge (\mathtt{PC} = 1)$

1 $\mapsto$ **2:** $(\mathtt{u'} = 2)$

2 $\mapsto$ **3:** $(\mathtt{u} \leq \mathtt{n}) \wedge (\mathtt{v'} = 1)$

2 $\mapsto$ **6:** $(\mathtt{u} = \mathtt{n} + 1)$

3 $\mapsto$ **2:** $(\mathtt{v} = \mathtt{u}) \wedge (\mathtt{u'} = \mathtt{u} + 1)$

3 $\mapsto$ **4:** $(\mathtt{v} \leq \mathtt{u} - 1)$

4 $\mapsto$ **5:** $\mathtt{G(u,v)} \wedge (\mathtt{Color[u]} = \mathtt{Color[v]})$

4 $\mapsto$ **3:** $\mathtt{G(u,v)} \wedge (\mathtt{Color[u]} \neq \mathtt{Color[v]}) \wedge (\mathtt{v'} = \mathtt{v} + 1)$

5 $\mapsto$ **5:** $(\mathtt{ack'} = \mathit{true}) \wedge (\mathtt{ret'} = \mathit{false})$

6 $\mapsto$ **6:** $(\mathtt{ack'} = \mathit{true}) \wedge (\mathtt{ret'} = \mathit{true})$

When only G is abstracted, the operator $\mathcal{A}$ transforms only the transitions **3** $\mapsto$ **4** and **3** $\mapsto$ **2** into:

(4 $\mapsto$ **5)':** $\big(\exists H,\ (H_\pi = \mathtt{G}_\pi) \wedge H(\mathtt{u},\mathtt{v})\big) \wedge (\mathtt{Color[u]} = \mathtt{Color[v]})$

(4 $\mapsto$ **3)':** $\big(\exists H,\ (H_\pi = \mathtt{G}_\pi) \wedge H(\mathtt{u},\mathtt{v})\big) \wedge (\mathtt{Color[u]} \neq \mathtt{Color[v]}) \wedge (\mathtt{v'} = \mathtt{v} + 1)$

Fix some n-vertex graph $G = (V, E)$. Suppose there is an accepting path in the abstracted transition system when the graph input is set to G, that is, the program variable G takes the value G. Along the path, when the program counter is **4**, there always exists a graph H such that $H_\pi = G_\pi$ and the system makes the transition **(4** $\mapsto$ **3)'**. Let $G_0 = (V, E_0)$ be the n-vertex graph whose vertices are defined by

$$(u, v) \in E_0 \quad \Longleftrightarrow \quad (u, v) \in E \text{ and } u, v \in \pi.$$

Observe that the transition is also made when the input is G_0. Thus there is an accepting path in the concrete system when the input graph is G_0.

In the general case, let us consider again an accepting path in the completely abstracted transition system. Again, one can prove that the abstracted transition **4** $\mapsto$ **3** is still made when the input is G_0. Then observe that only the indices of Color in π are relevant for this transition. Therefore, one can fix any value Color, such that $\mathit{Color}_\pi = \mathtt{Color}_\pi$. Thus the path of the concrete model which starts from the initial state $\mathtt{G} = G_0$ and $\mathtt{Color} = \mathit{Color}$, is again an accepting path. □

B Proofs of Lemmas 3 and 5

The proof of Lemma 3 hinges on Lemma 6, whose proof is based on Szemeredi's regularity lemma [14].

Lemma 6 ([10]). *Fix any complete graph on n vertices, with half the edges colored red and half the edges colored yellow. Then if n is large enough, there are disjoint sets of vertices A, B, C, such that the size of B*

is $\Omega(n)$, and there is a set of red (resp. yellow) pairwise vertex-disjoint paths $\{p_{x,y}|x,y \in B\}$ in $G_{R,Y}|_{A\cup B}$ (resp. $G_{R,Y}|_{B\cup C}$), where $p_{x,y}$ is a path of length 4 from x to y.

Proof of Lemma 3. For any coloring of the input variables, we construct a family of *HMT graphs*. Let A, B, C be the sets obtained from Lemma 6. Fix a pair of partitions P, P' of the vertices of B. Then for any pair x, y in the same part of P, include in $G_{P,P'}$ the edges in the red path of length 4 that is guaranteed to exist by Lemma 6. Do the same for the yellow paths that correspond to pairs of vertices in the same part of P'. Include no other edges. This completes the construction of $G_{P,P'}$. □

Proof of Lemma 5. Let P be a partition of the set $[m]$. The expression $P + l$ denotes the partition of $\{l+1, \ldots, m+l\}$ obtained from P by adding l to each element in all parts of P.

We consider only the partitions of the form $\{\{1\}, \ldots, \{2\sqrt{k+1}\}\} \cup (P^* + 2\sqrt{k+1})$ where P^* is a partition of $[n - 2\sqrt{k+1}]$. They will correspond to distinct lines in M.

Consider two distinct partitions $P_1 = \{\{1\}, \ldots, \{2\sqrt{k+1}\}\} \cup (P_1^* + 2\sqrt{k+1})$ and $P_2 = \{\{1\}, \ldots, \{2\sqrt{k+1}\}\} \cup (P_2^* + 2\sqrt{k+1})$ Since $P_1 \neq P_2$, there must be some pair x, y such that (without loss of generality) x, y are in the same part of P_1^*, but are in different parts in P_2^*.

Let $EVEN_k = \cup_{0\leq i<2\sqrt{k+1}}\{2i+2\}$ and $ODD_k = \cup_{0\leq i<2\sqrt{k+1}}\{2i+1\}$. Consider the partition P that contains parts $EVEN_k \cup \{x\}, ODD_k \cup \{y\}$ plus all the remaining vertices in singletons.

Now $H^k_{P_1,P}$ contains no edges, because x, y are in the same part of P_1^*. On the other hand, $H^k_{P_2,P}$ contains $k+1$ edges (counting multiple edges), because x, y are in different parts in P_2^*. Therefore the lines corresponding to P_1 and P_2 are different at column P.

The number of distinct lines must be at least the number of partitions of $[n - 2\sqrt{k+1}]$. This concludes the proof of the lemma. □

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'