Electronic Colloquium on Computational Complexity, Revision 1 of 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 which 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 within a particular input to decide if this input has a given property. Our approach brings the notion of sampling and approximation from property testing to model checking.

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 program with bounded inputs, modeled as a transition system, satisfies a specification, expressed as a formula of a temporal logic such as CTL [8]. This verification can be carried out in time linear in the number of states in the transition system [6]. 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 [14, 8] addresses this problem by using ordered binary decision diagrams [3, 4] (OBDDs, or equivalently read-once branching programs with an ordering restriction on the variables), which in many practical cases provide a compact representation of the transition system. Nevertheless, in some cases, such as integer multiplication and bipartiteness, the OBDD size remains exponential.

The abstraction method [7] provides a solution in some cases when the OBDD size is intractable. A large transition system is approximated by a smaller one, by way of an abstraction, and the spec-

[*]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

ification 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.

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 [5]. 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 [15] first formulated the notion of *property testing* which arises in every such tester. 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 a sublinear or even a constant number of queries.

Goldreich, Goldwasser, and Ron [9, 10, 11] 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 identify a notion which 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. Our goal will be to distinguish with sufficient confidence between programs that accept only graphs that satisfy ϕ and those which accept some graph that is ε-far from any graph that satisfies ϕ. We introduce *probabilistic abstractions* which to a program associate small random transition systems. We show that for probabilistic abstractions based on ε-reducibility this goal can be achieved.

In Section 2 we review basic notions of model checking and property testing, and define ε-reducibility (**Definition 6**). In Section 3, we introduce the notion of probabilistic abstraction (**Definition 7**). To be useful, an abstraction must preserve the behavior of the program. An abstraction is *congruent* (**Definition 9**) if it preserves program correctness. It is *robust* (**Definition 8** if correctness on the abstraction implies that the original program does not accept any graph that is ε-far from any graph that satisfies ϕ. The latter is an extension to abstraction of robustness introduced in [15].

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 applicability of our method by applying it to a program for bipartiteness. On the one hand, we show that a program for testing bipartiteness can be abstracted by our methods (**Corollary 1**). 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 small enough OBDDs. We prove, using methods from communication complexity, that the OBDD size remains exponential for approximate bipartiteness (**Theorem 7**).

2 Framework and preliminaries

2.1 Model checking framework

The goal of model checking is to automatize the process of verifying that a program's code is in agreement with formally stated specifications. We provide some background on how model checking achieves this goal. Briefly, *programs* are transformed into *transition systems* over an appropriate set of states, which represent the current value of all the program's variables at any given time.

On the other hand, a *specification* is given in a temporal logic, expressing a desired behavior of the program. The transition system determines the model in which the specification is interpreted. The problem of verifying the specification on the model, that is, the expression "model $\models$ specification", reduces to a simple decision procedure on OBDDs derived from the expression. The entire process can be carried out in polynomial time, provided the final reduction does not yield intractably large OBDDs. In the case of some problems, this is inherently unavoidable. An important issue in model checking is to devise ways around these large, in fact, exponential-size, OBDDs.

2.1.1 Programs and transition systems

We will use transition systems for representing all possible executions of a given program. Formally, a transition system is a triple defined as follows.

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

To simplify the discussion we will only consider programs which implement boolean functions. They will be written in a simple language that manipulates bit, bounded integer and finite array variables, using basic instructions: while statements, conditionals, assignments and a `get` instruction which allows the user to interact with the program. The *input variables* correspond to the input of the function. An implicit variable `ack` is set to *false* at the beginning of the program and is set to *true* at the end of the computation. Another implicit variable `ret` is defined together with the instruction `RETURN` such that `RETURN b` sets `ret` to `b` `ack` to *true*. To verify properties on the behavior of a program, we must know values of the variables at certain points of the program, called *control points*. The control points are at the beginning of lines labeled by integers. Finally, the implicit variable `PC` contains the label value of the last visited control point.

Suppose now that P is such a program with a finite set of variables $\{v_1, \ldots, v_n\}$ including the implicit variables `PC`, `ack`, and `ret`. Each variable v_i ranges over a (finite) domain D_i. We define the *transition system of P*. A *state of P* is an n-tuple $s = (s_1, \ldots, s_n)$ corresponding to a possible assignment of variables $v_1, \ldots, v_n$ at a control point during a computation. Then the *set of states of P* is $S = D_1 \times \ldots \times D_n$. The *initial states of P* are all the possible states before any computation starts, and the *transition relation of P* is the set of all possible transitions of the program between two control points. When the program terminates, the transition system loops with an infinite sequence of transitions on the final state.

Model checking does not manipulate the transition system directly; instead, it manipulates a logical representation of the transition system, expressed as a set of relational expressions. A *relational expression* is a formula of the first-order logic built up from the programming language's constants and basic operators (such as $+$, $-$, and $=$). Moreover we always assume that relational expressions are in negative normal form (negations pushed down to the atomic level).

Definition 2. *Let $\mathcal{I}$ (resp. $\mathcal{R}$) be a relational expression on S (resp. $S \times S$). Then $(\mathcal{I}, \mathcal{R})$ is a* representation of *$M = \langle S, I, R\rangle$ iff $I = \{s \in S : \mathcal{I}(s) = true\}$ and $R = \{(s, s') \in S \times S : \mathcal{R}(s, s') = true\}$.*

Let us now give an example.

```
FUNCTION GUESS
  INPUT a : BOOLEAN
  VAR b : BOOLEAN
1: get(b)
2: IF (a=b) RETURN true
   ELSE     RETURN false
```

The program variables are `a` and `b` with the implicit variables `PC`, `ack`, and `ret`. A state of the program is a 5-tuple $(\mathtt{PC}, \mathtt{ack}, \mathtt{ret}, \mathtt{a}, \mathtt{b})$. A transition of the program is a pair of states $\big((\mathtt{PC}, \mathtt{ack}, \mathtt{ret}, \mathtt{a}, \mathtt{b}), (\mathtt{PC}', \mathtt{ack}', \mathtt{ret}', \mathtt{a}', \mathtt{b}')\big)$. The relational expression for the initial states of the program is $(\mathtt{PC} = 1) \wedge (\mathtt{ack} = \mathit{false})$. The relational expression for the transition relation of the program is defined as the disjunction of the following three formulas:

$(\mathtt{PC} = 1) \wedge (\mathtt{PC}' = 2) \wedge (\mathtt{ack}' = \mathtt{ack}) \wedge (\mathtt{ret}' = \mathtt{ret}) \wedge (\mathtt{a}' = a)$,

$(\mathtt{PC} = 2) \wedge (\mathtt{PC}' = 2) \wedge (\mathtt{ack}' = \mathit{true}) \wedge (\mathtt{ret}' = \mathit{true}) \wedge (a = b) \wedge (\mathtt{a}' = \mathtt{a}) \wedge (\mathtt{b}' = \mathtt{b})$,

$(\mathtt{PC} = 2) \wedge (\mathtt{PC}' = 2) \wedge (\mathtt{ack}' = \mathit{true}) \wedge (\mathtt{ret}' = \mathit{false}) \wedge (a \neq b) \wedge (\mathtt{a}' = \mathtt{a}) \wedge (\mathtt{b}' = \mathtt{b})$.

Due to user interaction, $\mathtt{b}'$ does not appear in the first formula, and the first transition is therefore nondeterministic.

2.1.2 Temporal logic

To express the desired behavior of a transition system (associated to a program), we need a branching-time temporal logic. All our results will be stated for the temporal logic CTL*, but we only define the simplest fragment CTL and we refer the reader to [8] for more details on CTL*.

Formulas of CTL are defined inductively from a set of atomic propositions and built up by boolean connectives ($\neg$, $\wedge$, $\vee$), path quantifiers $\forall$ ("for all paths") and $\exists$ ("for some path"), temporal operators $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"). In our framework, atomic propositions are $(v_i = d)$ where v_i is a variable which corresponds to the ith coordinate of a state and d is any constant.

Definition 3. CTL *formulas are inductively defined as follows:*

1. *atomic propositions are* CTL *formulas;*
2. *boolean combinations of* CTL *formulas are* CTL *formulas;*
3. *if* Φ *and* Ψ *are* CTL *formulas then* $\forall(\mathbf{X}\,\Phi)$, $\exists(\mathbf{X}\,\Phi)$, $\forall(\Phi\,\mathbf{U}\,\Psi)$, $\exists(\Phi\,\mathbf{U}\,\Psi)$ *are* CTL *formulas.*

We now define the semantics of a CTL formula Θ for a transition system $M = \langle S, I, R\rangle$, with $S = D_1 \times \ldots \times D_n$. A *path* of the 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 $i \geq 0$. For every $s \in S$, we now inductively define $M, s \models \Theta$, that is *the satisfaction of a* CTL *formula* Θ *in* M *at state* s by:

1. $M, s \models (v_i = d_i)$ iff $s_i = d_i$ with $d_i \in D_i$.
2. Boolean connectives $\neg$, $\wedge$ and $\vee$ are defined as usual.
3. $M, s \models \forall(\mathbf{X}\,\Phi)$ (resp. $M, s \models \exists(\mathbf{X}\,\Phi)$) iff $M, \sigma_1 \models \Phi$ for all paths (resp. for some path) σ s.t. $\sigma_0 = s$.
4. $M, s \models \forall(\Phi\,\mathbf{U}\,\Psi)$ (resp. $M, s \models \exists(\Phi\,\mathbf{U}\,\Psi)$) iff for all paths (resp. for some path) σ s.t. $\sigma_0 = s$, $M, \sigma_k \models \Psi$ for some k, and $M, \sigma_j \models \Phi$ for all $0 \leq j < k$.

The notation $M \models \Theta$ means that $M, s \models \Theta$ for every initial state $s \in I$.

For example, the following CTL formula is a specification of the behavior of the program GUESS (see Section 2.1.1): $M \models \forall\Big(\neg\mathtt{ack}\,\mathbf{U}\,\mathtt{ack} \wedge \Big(\big(\mathtt{ret} \wedge (\mathtt{a} = \mathtt{b})\big) \vee \big(\neg\mathtt{ret} \wedge (\mathtt{a} \neq \mathtt{b})\big)\Big)\Big)$

2.1.3 OBDDs and model checking

We now focus on the verification of $M \models \Theta$. This is the heart of model checking. Although the standard model checking algorithm is linear in the size of the transition system M, the number of states in M is often too large to handle. For some problems, OBDD representation provides a succinct symbolic representation of the expression $M \models \Theta$. OBDDs were introduced by Bryant [3, 4] and can be defined as read-once branching programs, where a single variable is read at each level of the branching program.

Let us now briefly review the symbolic model checking method [8]. The verification process is unassisted, in that it takes as input the program and the specification, and decides whether the specification is verified. It is implemented by a procedure check which takes as arguments the CTL* formula and an OBDD representation of the transition relation of the transition system to be checked. The procedure returns an OBDD check(M, Θ) whose entries are states of S, and it satisfies for every $s \in S$

$$\text{check}(M, \Theta)(s) = \mathit{true} \iff M, s \models \Theta.$$

An OBDD is also constructed for $\mathcal{I}$. Classical logical operations on OBDDs (for instance, constructing an OBDD that computes the disjunction of two given OBDDs) can be carried out in quadratic time. Therefore, one can easily verify that $\big(\neg\mathcal{I} \vee \text{check}(M, \Theta)\big)$ is a tautology, that is, $M \models \Theta$ is true.

Thus, when the OBDD representing $M \models \Theta$ is polynomial in size, the verification can be carried out in polynomial time. A typical example where this is not the case is multiplication, since any OBDD for multiplication has exponential size [3]. In the next section, we will see how abstraction can overcome this problem.

2.1.4 Abstractions

The use of an *abstraction* [7] helps in some cases to overcome the problem of intractably large OBDDs. 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 4. *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 surjection $h : S \to \widehat{S}$, such that h can be decomposed into an n-tuple $h = (h_1, \ldots, h_n)$, where $h_i : D_i \to \widehat{D}_i$ is any surjection, and $\widehat{D}_i$ is any set. The* minimal transition system of M with respect to *h is the transition system $\widehat{M}_{\min} = \langle \widehat{S}, \widehat{I}_{\min}, \widehat{R}_{\min}\rangle$ such that $\widehat{S} = \widehat{D}_1 \times \ldots \times \widehat{D}_n$, $\widehat{I}_{\min} = h(I)$, and*

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

Note that the minimal transition system notion and all the notions that will follow are defined with respect to a fixed abstraction h. When we manipulate several abstractions together, we will specify the abstraction h as a superscript.

One can also define the minimal transition system directly from one of its representation. Define the $[\cdot]$ operator for every abstraction h and for any first order formula ϕ such that:

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

Then observe that $\widehat{R}_{\min} = [R]$.

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 its representation. Let us first define the notion of approximation.

Definition 5. *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 $\widehat{I}_{\min} \subseteq \widehat{I}$ and $\widehat{R}_{\min} \subseteq \widehat{R}$.*

To define an approximation directly on a representation of M, we simply apply $[\cdot]$ at the atomic level, *i.e.* to atomic relations and their negations. The obtained operator is named the *approximation operator* $\mathcal{A}$. More precisely, we define $\mathcal{A}$ inductively on formulas that are in negative normal form (negations pushed down to the atomic level) such that for every atomic proposition α, and first order formulas ϕ, ψ:

$$\begin{array}{ll} \mathcal{A}(\alpha) = [\alpha], & \mathcal{A}(\neg\alpha) = [\neg\alpha], \\ \mathcal{A}(\phi \wedge \psi) = \mathcal{A}(\phi) \wedge \mathcal{A}(\psi), & \mathcal{A}(\phi \vee \psi) = \mathcal{A}(\phi) \vee \mathcal{A}(\psi), \\ \mathcal{A}(\exists x\, \phi) = \exists \hat{x}\, \mathcal{A}(\phi), & \mathcal{A}(\forall x\, \phi) = \forall \hat{x}\, \mathcal{A}(\phi). \end{array}$$

For every transition system $M = \langle S, I, R \rangle$ with the representation $(\mathcal{I}, \mathcal{R})$, we will denote by $\mathcal{A}(M)$ the transition system with the set of states $\widehat{S} = h(S)$ and with the representation $(\mathcal{A}(\mathcal{I}), \mathcal{A}(\mathcal{R}))$. Then the following result states that the approximation operator correctly defines approximated transition systems.

Theorem 1 **([7]).** *Let $M = \langle S, I, R \rangle$ be a transition system with any representation $(\mathcal{I}, \mathcal{R})$, and let $h : S \to \widehat{S}$ be an abstraction for M. Then the transition system $\mathcal{A}(M)$ approximates M with respect to h.*

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 transformations $\mathcal{C}$ and $\mathcal{D}$ between formulas on M and their approximation on $\widehat{M}$. These transformations preserve boolean connectives, path quantifiers, and temporal operators, and act on atomic propositions as follows:

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

Denote by $\forall$CTL* and $\exists$CTL* the universal fragment and the existential fragment of CTL*. Then the following theorem states correspondences between concrete models and their approximations.

Theorem 2 **([7]).** *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 $\forall$CTL* formula on $\widehat{M}$, and Θ' be a $\exists$CTL* formula on M. Then*

$$\widehat{M} \models \Theta \implies M \models \mathcal{C}(\Theta), \qquad \textit{and} \qquad M \models \Theta' \implies \widehat{M} \models \mathcal{D}(\Theta').$$

The second part of the theorem is implicit in [7]. Notice that the two statements are not reciprocals of one another because of the type of formulas involved ($\forall$CTL* and $\exists$CTL*). In both cases, reciprocals can be shown under certain conditions on the abstractions [7]. The first result validates the usefulness of abstractions in practical model checking. The second will be used in our proof of Theorem 5.

2.2 Property testing

In what follows, we will use property testing to obtain a new family of abstractions for model checking. We now outline the basic notions pertaining to 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 number of edges on which the graphs disagree divided by n^2. Nevertheless, our theory and our main results (Theorems 5 and 6) 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 graph with property ϕ, and rejects with probability 2/3 every graph which has distance more than ε from any graph having the property.[1]

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

Moreover, an ε-test can only access the input graph by querying whether or not any chosen pair of vertices are adjacent. 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 a 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 6. *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,*

$$\begin{aligned} G \models \phi \quad &\Longrightarrow \quad \forall \pi \in \Pi,\ G_\pi \models \psi, \\ G \not\models \phi_\varepsilon \quad &\Longrightarrow \quad \mathop{\mathbf{Pr}}_{\pi \in \Pi}[G_\pi \models \psi] \leq 1/3. \end{aligned}$$

Note that the expression $G \not\models \phi_\varepsilon$ means that G has distance more than ε from any graph satisfying ϕ.

For example, we can recast the results of k-colorability and bipartiteness [9] in terms of ε-reducibility.

Theorem 3 ([9]). *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 bound variables.*

3 Verification of graph properties

3.1 Context and objectives

Our goal is to extend the framework of model checking to include the use of probabilistic abstractions. In order to do so we would like to prove an analogue of Theorem 2. We prove that with high probability, incorrect programs will be rejected (*ε-robustness*). We also prove a reciprocal, which states that under a certain condition (*exactness*), correct programs will never be rejected (*congruence*).

A second goal is to extend the framework of model checking to include the verification of programs purportedly deciding graph properties. The standard model checking method is not adapted to programs on inputs that are first-order structures such as graph adjacency relations. We overcome this by dealing with the specification of the program, and the property of the graph, separately. The former is handled with standard tools of model checking. The latter will reduce, as a result of the ε-reduction, to verifying a property on constant size graphs, which can be carried out in constant time.

We give an example of a program for bipartiteness, and show that the approximation operator $\mathcal{A}$ results in an exact approximation of the transition system. Hence, the ε-abstraction can be used to verify the program. Finally, one might ask whether the relaxation brought about by the use of property testing is in itself enough to beat the exponential lower bounds on the original problem. We show that this is not the case, by giving a lower bound on the relaxed version of bipartiteness.

Consider the following special case of a formula we wish to verify, where P is a program which computes some boolean function on bounded size graphs:

The program P accepts only graphs which satisfy some graph property ϕ.

Suppose that $\mathtt{G}$ is an input variable of P, such that $\mathtt{G}$ is interpreted as a graph G (with respect to some fixed encoding): this will be written as $\mathtt{G} = G$.

A state s of the transition system $M = \langle S, I, R\rangle$ of P is a finite sequence of variables $(\ldots, \mathtt{G}, \ldots)$. For every graph G, we then define $I_G = \{s \in I : \mathtt{G} = G\}$. Let ϕ be some property. We would like to check the following:

$$\forall G \quad \Big((\forall s \in I_G \quad M, s \models \exists\ ((\neg \mathtt{ack})\, \mathbf{U} (\mathtt{ack} \wedge \mathtt{ret}))) \quad \Longrightarrow \quad G \models \phi \Big).$$

Note that on the RHS of the implication, ϕ is interpreted in a structure for G which does not include the transition system. This is because the standard model checking algorithms are not suited for programs with inputs that are first order structures. When there is no ambiguity, we will write $M, G \models \Theta$ instead of $\forall s \in I_G, \quad M, s \models \Theta$.

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 includes the graph G and may also include auxiliary data, Θ is a CTL* formula, and ϕ is a graph property. In the remainder of the paper, we always assume a graph G to be an input variable in the program.

Since G is a bounded size graph and ϕ is a formula expressing a property on G, we can determine whether $G \models \phi$ using an OBDD. Let $\mathrm{sat}(\phi, G)$ be such an OBDD. Then verifying (1) can be achieved by checking the validity of $\Big((\neg \mathcal{I}_G \vee \mathrm{check}(M, \Theta)) \implies \mathrm{sat}(\phi, G) \Big)$, where $\mathcal{I}_G = \mathcal{I}(G/\mathtt{G})$ (all occurrences of the variable $\mathtt{G}$ are substituted for G).

For the graph properties that we consider, such as bipartiteness, the OBDDs for $G \models \phi$ have exponential size. As we show in Section 4, the relaxation of property testing is not sufficient to reduce the OBDD size of bipartiteness. We use ε-reducibility to construct probabilistic abstractions, yielding smaller, even constant-size OBDDs. Using these small OBDDs, we are able to guarantee that P approximately decides ϕ on all its inputs.

3.2 Probabilistic abstraction

Definition 7. *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 $h \in \mathcal{H}$ to 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 we require that $\widehat{G}^h$ be interpreted as a graph, and the operator $\mathcal{D}$ (see Section 2.1.4) is applied with respect to the appropriate abstraction.

```
Generic Test$((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi)$
    1. Choose an element $h \in \mathcal{H}$ according to $\mu$.
    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 $\mathsf{Rej}((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi)$. The distribution μ will be omitted when it denotes the uniform probability distribution. To be useful in practice, a probabilistic abstraction should be both ε-robust (programs are rejected with probability 2/3 if the relaxed specification is false for some input) and congruent (no correct programs are rejected), in which case we say that it is an ε-abstraction. Then checking the correctness of a program can be easily done on the abstracted model with high confidence using **Generic Test**. Fix a confidence parameter $0 < \gamma < 1$, and iterate **Generic Test** $O(\ln 1/\gamma)$ times. If the program is correct, **Generic Test** always accepts; and if there is an instance on which the program is not correct with respect to the relaxed specification, **Generic Test** rejects at least once with probability greater than $(1-\gamma)$.

Definition 8. *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*

$$(\exists G \quad (M, G \models \Theta \quad \text{and} \quad G \not\models \phi_\varepsilon)) \quad \Longrightarrow \quad \mathsf{Rej}\big((\mathcal{H}, \mathcal{M}, \mu), \Theta, \psi\big) \geq \tfrac{2}{3}.$$

Definition 9. *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 10. *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. From Definition 6, for any fixed k, we let Π be the set of all subsets π of vertices with $|\pi| = k$, and for any graph G, the vertex-induced subgraph on the vertex set π is denoted by G_π.

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. Every corresponding abstracted variable v and constant d will be denoted respectively by $\widehat{v}^\pi$ and $\widehat{d}^\pi$. 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((Π, 𝓜), Θ, ψ)
    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 $\exists$CTL* formula, then our probabilistic abstraction is ε-robust. This, together with its conditional reciprocal in Theorem 6, establishes the validity of the method.

Theorem 5. *Let Θ be a $\exists$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 $G \not\models \phi_\varepsilon$. 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: exactness.

Definition 11. *Let M be a transition system, Θ be a* CTL* *formula, 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 $\exists$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. We consider a function which decides, given a graph G and a coloring *Color* (which is captured by the user), if *Color* is a bipartition for G. The graph G is represented by the upper triangular entries of a matrix `G` and *Color* by an array `Color`.

```
FUNCTION CHECK-PARTITION
  CONSTANT INTEGER n=10000
  INPUT G : ARRAY[n,n] of BOOLEAN
  VAR Color : ARRAY[n] of BOOLEAN
  VAR u,v : INTEGER 1..n+1

1: get(Color)
2: u=2
3: WHILE u<=n DO {
     v=1
4:   WHILE v<=u-1 DO {
5:     IF G[u,v] && (Color[u]=Color[v]) 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 value for `Color` for which the program accepts, then `G` represents a bipartite graph. More formally, we want to verify the following property:

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

Note that $\exists$ ranges over all the possible initial values of `Color` which the user can enter with the instruction `get(Color)`. For each π we define the abstraction which maps G to the subgraph G_{π}, *Color* to the coloring on the subset of vertices induced by π, and `u`, `v` refer to the vertices as follows:

$$\widehat{u}^{\pi} = \begin{cases} u, & \text{if } u \in \pi, \\ \min\{w : \ \forall t\ (w \leq t \leq u \implies t \notin \pi)\}, & \text{otherwise.} \end{cases}$$

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

Lemma 1. *For every $\pi \in \Pi$, $\mathcal{A}^{\pi}(M)$ is exact with respect to the* $\exists$CTL* *formula in* (2).

Since the size of π is fixed, our abstraction induces a constant size OBDD. By Lemma 1, 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).

Corollary 1. *Let $\varepsilon > 0$. Using previous probabilistic abstraction,* **Graph Test** *on* `CHECK-PARTITION` *satisfies:*

1. *If* `CHECK-PARTITION` *satisfies specification* (2), **Graph Test** *always accepts;*
2. *If there exists a graph G which has distance more than ε from any bipartite graph, but which is accepted by* `CHECK-PARTITION` *for some coloring Color, then* **Graph Test** *rejects with probability at least* $2/3$*;*
3. *The time complexity of* **Graph Test** *is polynomial in* $1/\varepsilon$ *and does not depend on* n, *the input size.*

4 Lower bound for approximate bipartiteness

In this section, we show how the communication complexity lower bound of Hajnal, Maass, and Turán [12] 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 arguments.

A graph $G = (V, E)$ is *k-bipartite* if there is a set of edges $F \subseteq E$ with $|F| \leq k$ such that $G' = (V, E \setminus F)$ is bipartite. In particular, a graph is 0-bipartite if and only if it is bipartite. The k-bipartiteness is the following partially defined problem.

Definition 12 (k-bipartiteness). *Let k be an integer. The k-bipartiteness problem is a partial function f on graphs G:*

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

We say that an OBDD *solves* the k-bipartiteness problem if its output agrees with f whenever f is defined. The rest of this section is devoted to proving the following theorem.

Theorem 7. *Any OBDD solving the k-bipartiteness 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* $2^{\Omega((1-2\sqrt{\varepsilon})n\log((1-2\sqrt{\varepsilon})n))}$.

4.1 Preliminaries

We denote by $S\dot{\cup}T$ the disjoint union of sets S and T. A *partition* of a finite set S is a set of non-empty parts $S_1, S_2, \ldots$ 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 $2^{\Omega(n\log n)}$.

Two kinds of partitions will be considered: 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 (plus or minus one).

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* $\kappa_{\rightarrow}^{R:Y}(\mathcal{P}; x, y)$ *incurred by a one way-communication protocol* $\mathcal{P}$ 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 $\kappa_{\rightarrow}^{R:Y}(f) = \min_{\mathcal{P}} \max_{x,y}\{\kappa_{\rightarrow}^{R:Y}(\mathcal{P}; x, y)\}$. The *one-way communication complexity for the best-case coloring of variables* is $\kappa_{\rightarrow}^{best}(f) = \min_{R\dot{\cup}Y}\{\kappa_{\rightarrow}^{R:Y}(f)\}$.

Let f be a boolean function whose variables are colored by R, Y. The *communication matrix associated with f* is the matrix representation M_f of f, that is $M_{f,x,y} = f(x, y)$. The lower bound on the width of OBDDs that compute f is related to the communication matrix of f by the following proposition, proven in [13, Page 144] for general communication complexity, in terms of the rank of the communication matrix. We state the result for one-way communication complexity, and require only a lower bound on the number of distinct lines of the matrix.

Proposition 1 (follows from [13, Page 144]).

1. *If f has an OBDD of width at most w, then* $\kappa_{\rightarrow}^{best}(f) \leq \log w$.
2. *Let M_f be the communication matrix associated with the boolean function f whose variables are colored by R, Y. Then $\kappa_{\rightarrow}^{R:Y}(f) \geq log(l)$, where l is the number of distinct lines in M_f.*

Since we will study partially defined problems the communication matrices we consider will have entries 0 and 1 when the problem is defined, 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.

4.2 The k-partition problem

We introduce the k-partition problem. In Section 4.3 we will show that it appears as a submatrix of the k-bipartiteness communication matrix.

We denote by $[m]$ the set $\{1, \dots, m\}$. In the rest of the paper, we fix for every integer $k \geq 1$, a set E_k of $k+1$ bipartite edges from vertices of $[2\lceil\sqrt{k+1}\rceil]$ in such a way that these new edges only go between even and odd vertices.

Definition 13 (Figure 1 in Appendix B). *Let $k, m \geq 0$ be two integers. For any partition P of $[m]$, H_P^k denotes the multigraph (where multiple edges are allowed) such that*

1. *vertices are the parts of P,*
2. *there is an edge in H_P^k between two parts $Q, Q' \in P$ if and only if some edge $e \in E_k$ crosses over Q, Q', that is, $e = \{a, b\}$, with $a \in Q$ and $b \in Q'$.*

These graphs lead to the following partially defined problem.

Definition 14 (k-partition problem). *Let $k, m \geq 0$ be two integers. The k-partition problem is a partial function g on partitions P of $[m]$:*

$$g(P) = \begin{cases} 1 & \text{when } H^k(P) \text{ contains } k+1 \text{ edges and no odd cycle,} \\ 0 & \text{when } H^k(P) \text{ is empty.} \end{cases}$$

For any two partitions P, P' of a set X, $P{\vee}P'$ is the finest partition which is refined both by P and P'. For k-partition problem, the input of Player I is (an encoding of) some partition P, and the input of Player II is (an encoding of) a partition P'. The goal of the communication game is to compute $g(P \vee P')$. The corresponding communication matrix is M, whose rows and columns are labeled by partitions P and P' of $[m]$. The number of rows and columns is B_m, the mth Bell number ($B_m \approx 2^{m \log m}$). We show that M has an exponential number of distinct lines.

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

Proof. Let P be a partition of the set S. The expression $P+l$ denotes the partition of $\{x+l | x \in S\}$ obtained from P by adding l to each element in all parts of P.

To show that the number of distinct lines is large, we exhibit a large subset of lines which are pairwise distinct. Consider the partitions of the form $\{\{1\},\ldots,\{2\sqrt{k+1}\}\} \cup (P^* + 2\sqrt{k+1})$ where P^* is a partition of $[m - 2\sqrt{k+1}]$. These will correspond to distinct lines in M.

Consider two such 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 $P_1 \neq P_2$ only on the "second part", 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^*. Further we will use the notation $Q_{x,y}$ to mean the part of P_1^* that contains x and y, and Q_x, Q_y to mean the parts of P_2^* that contain x, y, respectively.

We exhibit a partition P' such that $M_{P_1,P'}$ differs from $M_{P_2,P'}$. Recall that the new edges E_k connect only odd vertices to even vertices. Let $EVEN_k = \cup_{1 \leq i < \sqrt{k+1}}\{2i\}$ and $ODD_k = \cup_{1 \leq i < \sqrt{k+1}}\{2i-1\}$. Let P' contain the parts $EVEN_k \cup \{x\}, ODD_k \cup \{y\}$ plus all the remaining vertices in singletons. Then,

$$\begin{aligned} P_1 \vee P' &= EVEN \cup ODD \cup Q_{x,y}, \{2\sqrt{k+1}+1,\ldots,m\} \setminus Q_{x,y} \\ P_2 \vee P' &= EVEN \cup Q_x, ODD \cup Q_y, \{2\sqrt{k+1}+1,\ldots,m\} \setminus (Q_x \cup Q_y) \end{aligned}$$

Now $H^k_{P_1 \vee P'}$ contains no edges, because all the new edges go between odd and even vertices. Likewise, $H^k_{P_2 \vee P'}$ contains a single edge of multiplicity $k+1$. 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 P^* of $[m - 2\sqrt{k+1}]$. This concludes the proof of the lemma. □

4.3 Reduction to k-bipartiteness

In this section, we show that the communication matrix associated with the k-partition problem appears as a submatrix of the communication matrix of the k-bipartiteness problem (Lemma 4).

For the k-bipartiteness problem, the variables in the communication problem are pairs of vertices: the variable is 1 whenever the corresponding edge is in the graph, and 0 otherwise. Each player is given half of the variables partitioned according to some coloring. The input of the communication protocol is the graph formed by the union of edges from the players' variables.

Hajnal *et al.* give a reduction from bipartiteness to a property on partitions. We show how this reduction can be extended to show a lower bound on the relaxed version of bipartiteness by reducing k-bipartiteness to the k-partition problem. We re-use the main technical component of [12], namely, HMT graphs.

4.3.1 HMT graphs

The construction of [12] produces a large family of graphs (parameterized by P, P') that can be embedded into a coloring of the edge variables, 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 P' of the same set.

Definition 15 ([12], Figure 2 in Appendix B). *Let $n \geq 1$ be an integer, and $A\dot{\cup}B\dot{\cup}C \subseteq [n]$ with $|B| = \Omega(n)$. Let $\{G_{P,P'}\}$ be a graph family with vertices $[n]$, where P, P' ranges over the pairs of partitions of B. $\{G_{P,P'}\}$ is an* HMT graph family on A, B, C for a coloring R, Y *if for any pair of vertices $x, y \in B$ included in some part of P (resp. P'), the only edges of the graph $G_{P,P'}$ are vertex-disjoint red (resp. yellow) paths in $A \cup \{x, y\}$ (resp. $C \cup \{x, y\}$) of length* 4.

Lemma 3 ([12]). *Fix any coloring of the edge variable of graphs with vertices $[n]$, with half the edges colored red and half the edges colored yellow. Then if n is large enough, there exists sets A, B, C such that there is an HMT graph family on A, B, C for this coloring of the edge variables.*

4.3.2 Bipartiteness and partitions

For a coloring R, Y of the edge variables, we show how to construct a large family of instances of the k-partition problem from a family of HMT graphs for this coloring. A key observation is that the connected components of an HMT graph $G_{P,P'}$ correspond exactly to the parts of $P \vee P'$.

Proposition 2 ([12]). *Let $\{G_{P,P'}\}$ be an HMT graph family on A, B, C for a coloring R, Y. Let $v_1, v_2 \in B$ be two distinct vertices. For any pair of partitions P, P' of B, 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 $Q \in P \vee P'$.*

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 $G_{P,P'}$.

Without loss of generality, we now renumber the vertices so that $B = \{1, 2, \ldots\}$. For the k-bipartiteness problem, instead of adding a single, fixed, edge of $B \times B$ to each graph $G_{P,P'}$, we add the fixed bipartite set E_k of $k+1$ edges (see Section 4.2). We call the resulting graph $G^k_{P,P'}$.

There are two ways in which an odd cycle can be created in $G_{P,P'}$ 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 $H^k_{P \vee P'}$ (see Definition 13).

Lemma 4. *Let $\{G^k_{P,P'}\}$ be an HMT graph family on A, B, C for a coloring R, Y.*

1. *$G^k_{P,P'}$ is bipartite if and only if $H^k_{P \vee P'}$ has $k+1$ edges and no odd cycle.*
2. *$G^k_{P,P'}$ is not k-bipartite if and only if $H^k_{P \vee P'}$ has no edges.*

Proof. For the implication of Part 1, we show the contrapositive. If $H^k_{P \vee P'}$ has fewer than $k+1$ edges, then some edge in E_k lies within a part of $P \vee P'$. By definition of $G^k_{P,P'}$, this creates a cycle of length 5. If $H^k_{P \vee P'}$ contains an odd cycle of length t, then this forms a cycle in $G^k_{P,P'}$ of length t plus a multiple of 4. For the converse, notice that the new edges form a bipartite subgraph so they cannot form an odd cycle on their own.

For the implication of Part 2, we show the contrapositive. Assume that some edge $e \in E_k$ gave rise to an edge in $H^k_{P \vee P'}$. Removing k edges suffices to make $G^k_{P,P'}$ bipartite, because it is enough to remove all the new edges except e. For the converse, assume $H^k_{P \vee P'}$ is empty. By definition the paths in $G^k_{P,P'}$ are vertex disjoint. Furthermore, the $k+1$ new edges form a bipartite graph. Therefore $k+1$ edges must be removed from $G^k_{P,P'}$ to remove the $k+1$ odd cycles. □

Lemma 4 establishes that the k-partition communication matrix appears as a submatrix of the k-bipartiteness communication matrix. Theorem 7 therefore follows, by Proposition 1, from the lower bound on the number of distinct lines, proven in Lemma 2.

Acknowledgements

We would like to thank Lokam V. Satyanarayana for discussions on the proof in Section 4, and Miklos Santha for many comments on an earlier draft of the paper.

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] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, C-35(8):677–691, 1986.

[4] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318,1992.

[5] 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.

[6] 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.

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

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

[9] 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.

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

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

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

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

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

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

[16] E. Szemerédi. Regular partitions of graphs. In Éditions du CNRS, *Problèmes combinatoires et théorie des graphes*, pages 399–401, 1978.

A Proof of Lemma 1

Proof of Lemma 1. 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.

For convenience, we use a compact representation of relational expressions representing the transition relation. Each line corresponds to a transition between two control points. The transition relation is represented by the disjunction of these lines. We use '$i \mapsto j$:' as an abbreviation for $(\mathtt{PC} = i) \wedge (\mathtt{PC}' = j)$. On any given line, for any pair $(\mathtt{v}, \mathtt{v}')$ of program variables, if $\mathtt{v}'$ does not occur in the relational expression, then the atomic proposition $(\mathtt{v}' = \mathtt{v})$ is understood, but omitted from the compact form. Furthermore, the expression $(\mathtt{v}' = *)$ is used when the value of $\mathtt{v}'$ is unspecified. This typically occurs after a `get` instruction, and corresponds to a nondeterministic transition.

The initial states of the transition system of `CHECK-PARTITION` are $(\mathtt{ack} = \mathit{false}) \wedge (\mathtt{PC} = 1)$ The relational expression of the transition system of `CHECK-PARTITION` is given in compact form by the disjunction of the following boolean formulas.

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

$2 \mapsto 3 :\ (\mathtt{u'} = 2)$

$3 \mapsto 4 :\ (\mathtt{u} \leq \mathtt{n}) \wedge (\mathtt{v'} = 1)$

$3 \mapsto 6 :\ (\mathtt{u} = \mathtt{n} + 1)$

$4 \mapsto 3 :\ (\mathtt{v} = \mathtt{u}) \wedge (\mathtt{u'} = \mathtt{u} + 1)$

$4 \mapsto 5 :\ (\mathtt{v} \leq \mathtt{u} - 1)$

$5 \mapsto 5 :\ \mathtt{G(u,v)} \wedge (\mathtt{Color[u]} = \mathtt{Color[v]}) \wedge (\mathtt{ack'} = \mathit{true}) \wedge (\mathtt{ret'} = \mathit{false})$

$5 \mapsto 4 :\ \mathtt{G(u,v)} \wedge (\mathtt{Color[u]} \neq \mathtt{Color[v]}) \wedge (\mathtt{v'} = \mathtt{v} + 1)$

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

We first suppose that only `G` is abstracted. Then the operator $\mathcal{A}$ transforms only the relational expression part of transitions $5 \mapsto 5$ and $5 \mapsto 4$ into:

$5 \overset{\pi}{\mapsto} 5 :\ \big(\exists H\ (H_\pi = \mathtt{G}_\pi) \wedge H(\mathtt{u}, \mathtt{v})\big) \wedge (\mathtt{Color[u]} = \mathtt{Color[v]})$
$\quad \wedge (\mathtt{ack'} = \mathit{true}) \wedge (\mathtt{ret'} = \mathit{false})$

$5 \overset{\pi}{\mapsto} 4 :\ \big(\exists H\ (H_\pi = \mathtt{G}_\pi) \wedge H(\mathtt{u}, \mathtt{v})\big) \wedge (\mathtt{Color[u]} \neq \mathtt{Color[v]}) \wedge (\mathtt{v'} = \mathtt{v} + 1)$

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 5, there always exists a graph H such that $H_\pi = G_\pi$, so the system makes the transition $5 \overset{\pi}{\mapsto} 4$. 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 $5 \overset{\pi}{\mapsto} 4$ 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 $\widehat{\mathtt{Color}}^\pi = \widehat{\mathit{Color}}^\pi$. 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. $\square$

B Figures

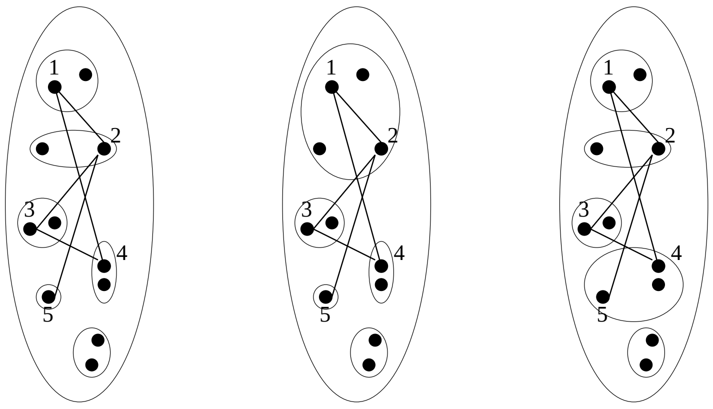

Figure 1: Three examples of H_P^k graphs.

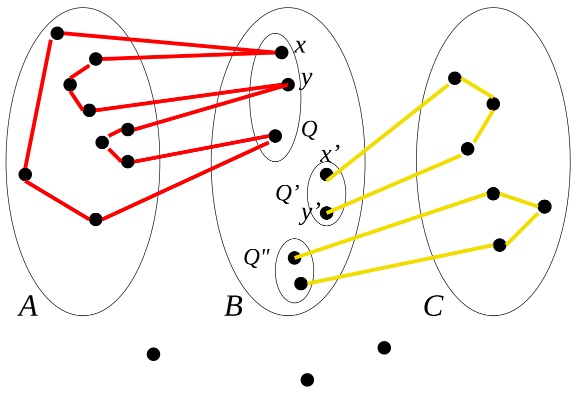

Figure 2: An HMT graph: $Q \in P$, $Q', Q'' \in P'$.