Oracle Circuits for Branching-Time Model Checking

Ph. Schnoebelen

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: phs@lsv.ens-cachan.fr

Abstract. A special class of oracle circuits with tree-vector form is introduced. It is shown that they can be evaluated in deterministic polynomial-time with a polylog number of adaptive queries to an NP oracle. This framework allows us to evaluate the precise computational complexity of model checking for some branching-time logics where it was known that the problem is NP-hard and coNP-hard.

Keywords: Branching-Time Temporal Logic, Model Checking, Logic of Programs, Oracle Machines.

1 Introduction

Many different *temporal logics* have been proposed in the computer science literature [Eme90]. Their main use is in the field of reactive systems, where *model checking* allows automated verification of correctness [CGP99].

Comparing and classifying the different temporal logics is an important task. This is usually done along several axis, most notably *expressive power* and *computational complexity*. Generally, in model checking applications, one looks for frameworks offering the best compromise between high expressive power and low model checking complexity.

Regarding expressive power, the principal questions have been answered [1] and it is more or less known how the main temporal logics fare [Eme90,Rab02].

Regarding computational complexity, the available knowledge is not so exhaustive [Sch03]. In particular, for several branching-time temporal logics (or some of their natural fragments), the complexity of model checking is not known [2]. Advances in this domain are welcome since it is important to understand what ideas underly "optimal" algorithms, and what special cases may benefit from specialized methods.

[1] Including the recent proof that linear-time temporal with past is at least exponentially more succinct than its pure-future fragment [LMS02b].

[2] There is a similar situation with the complexity of *satisfiability* (or equivalently validity) for branching-time logics, but algorithms for their satisfiability have less practical importance since they are usually not implemented in tools.

Model checking in the polynomial-time hierarchy. There is a family of branching-time temporal logics for which the complexity of model checking is not known precisely. These logics can be described as branching-time logics where the underlying path properties are in NP or coNP (we give several examples in section 2).

Let us write $B(L)$ for the branching-time extension of some linear-time logic L. If model checking for L is in NP (or, more likely, in coNP), then model checking $B(L)$ is in the polynomial-time hierarchy, inside the level P^{NP} (also called Δ_2^p). For logics like CTL^+ (i.e. $B(L^1(\mathsf{U},\mathsf{X}))$ or $B^*(\mathsf{F})$ (i.e. $B(L(\mathsf{F}))$), the Δ_2^p upper bound has been known for almost twenty years.

For such logics, the question of finding matching lower bounds saw no progress until recently, when Laroussinie, Markey, and the author managed to prove that some of them (including $B^*(\mathsf{F})$, CTL^+, and $FCTL$) have indeed a Δ_2^p-complete model checking problem [LMS01,LMS02a].

However, for some remaining logics, the techniques used in [LMS01,LMS02a] for proving Δ_2^p-hardness do not apply. The difficulty here is that, if these problems are *not* Δ_2^p-complete, we still lack methods for proving that a model checking problem has upper bounds higher than NP or coNP but lower than Δ_2^p.

Our contribution. In this paper we develop a framework that allows proving upper bounds below Δ_2^p and apply it to branching-time model checking problems. The approach is successful in that it allows us to prove model checking $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete, and model checking *Timed* $B(\mathsf{F})$ is $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-complete.

Our framework is based on Boolean circuits with oracle queries (introduced in [Wil87]). We identify two special classes of oracle circuits having *tree-vector* form (with special constraints on the oracle queries) for which we prove evaluation can be done in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and, respectively, $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$, i.e. they can be evaluated by a deterministic polynomial-time Turing machine that makes $O(\log n)$ (resp. $O(\log^2 n)$) adaptive queries to an NP-oracle (while Δ_2^p-complete problems require [3] polynomially-many adaptive queries).

Branching-time model checking problems lead naturally to tree-vector circuits, so that we obtain upper bounds directly by translations. The lower bounds are proved by ad-hoc reductions: once the complexity class we aim at has been properly identified, the main ingredient is the usual dose of ingenuity.

These results are important for several reasons:
1. The tree-vector oracle circuits may have more applications than just in model checking. In any case, they illuminate a structural feature of model checking where the formula is a modal expression *tree* evaluated over a *vector* of worlds.
2. The results help complete the picture in the classification of temporal logics. A logic like $B^*(\mathsf{X})$, the *full branching-time logic of "next"*, is perhaps not used in practice, but it is a fundamental fragment of CTL^*, for which we should be able to assess the complexity of model checking.

[3] That is, assuming Δ_2^p does not collapse to $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$ and $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$! We shall often write such sentences that implicitly assume the separation conjectures most complexity theorists believe are true.

3. They provide examples of problems complete for $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$. Very few such examples are known. In particular, with the model checking of $B^*(\mathsf{X})$, we provide the first example (to the best of our knowledge) of a natural problem complete for $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$.

Related work. The best known framework for assessing the complexity of model checking problems is the automata-theoretic framework initiated by Vardi and Wolper [VW86]. By moving to tree-automata, this framework is able to deal with branching-time logics [KVW00], where it has proved very successful (see the unexpected [DW99]). However, the tree-automata approach seems too coarse-grained for our problems where it seems we need a fine-grained look at the structure of the oracle calls.

Gottlob's work on NP trees [Got95] was an inspiration. His result prompted us to check whether certain tree-vectors of queries could be normalized (and then solved efficiently using the techniques of Castro and Seara [CS96]).

Plan of the paper. We recall the necessary background in Section 2. Then Section 3 is devoted to tree-vector oracle circuits and flattening algorithms for evaluating them. This lays the ground for our proof that model checking $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete (Section 4) and model checking *Timed* $B(\mathsf{F})$ is $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-complete (Section 5).

2 Branching-time logics with model checking in Δ_2^p

2.1 Complexity classes below Δ_2^p

We assume familiarity with computational complexity. The main definitions we need concern classes in the polynomial-time hierarchy (see [Joh90,Pap94]).

Δ_2^p is the class P^{NP} of problems solvable by deterministic polynomial-time Turing machines that may query an NP oracle. Some relevant subclasses of Δ_2^p have been identified:

- $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ (from [PZ83]) only allows $O(\log n)$ oracle queries instead of polynomially-many. For example, PARITY-SAT (the problem where one is asked whether the number of satisfiable Boolean formulae from some input set $f_1, \ldots, f_n$ is odd or even) is $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-complete [Wag87].
- $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ (from [BH91]) only allows one round of *parallel* queries: the polynomially-many queries may not be adaptive (i.e., depend on the outcomes of earlier oracle queries) but must first be all formulated *before* the oracle is consulted on all queries. Then the computation proceeds normally, using the polynomially-many oracle answers.

Buss and Hay showed that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ and $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ coincide (and they further coincide with $\mathrm{P}_{\parallel O(1)}^{\mathrm{NP}}$, where a fixed number of parallel rounds is allowed) [BH91]. Wagner showed that many different and natural ways of restricting Δ_2^p all lead to

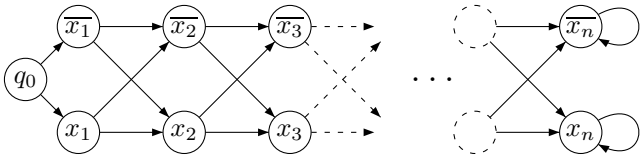
the same $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ class (e.g. $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ coincide with L^{NP}), for which he introduced the name Θ_2^p [Wag90]. Further variants were introduced by Castro and Seara, who proved that, for all $k \in \mathbb{N}$, $\mathrm{P}^{\mathrm{NP}[O(\log^k n)]}$ coincide with $\mathrm{P}^{\mathrm{NP}}_{\|O(\log^{k-1} n)}$ (where a succession of $O(\log^{k-1})$ parallel querying rounds are allowed) [CS96].

2.2 Branching-time logics and NP-hard path modalities

We assume familiarity with temporal logic model checking [Eme90,CGP99,Sch03]. Several branching-time logics combine the path quantifiers E and A with linear-time modalities whose path existence problem is in NP. Here are five examples:

- *FCTL* [EL87], or "Fair *CTL*", allows restricting to the fair paths of a Kripke structure, where the fair paths are defined by an arbitrary Boolean combination of $\overset{\infty}{\mathsf{F}} \pm P_i$s. The existence of a fair path is NP-complete [EL87].
- *TCTL* [Koy90], or "Timed *CTL*", allows adding timing subscripts to the usual modalities. In Timed KSs (i.e. Kripke structures where edges carry a discrete "duration" weight) the existence of a path of a given accumulated duration is NP-complete [LMS02a].
- CTL^+ [EH85] allows arbitrary Boolean combinations (not nesting) of the U and X modalities under a path quantifier. Thus CTL^+ is the branching-time extension of $L^1(\mathsf{U},\mathsf{X})$, the fragment of linear-time logic with modal depth one, for which the existence of a path is NP-complete [DS02].
- $B^*(\mathsf{F})$ and $B^*(\mathsf{X})$ are the branching-time extensions of $L(\mathsf{F})$ and $L(\mathsf{X})$ (resp.). $B^*(\mathsf{F})$ (called BT^* in [CES86]) is the *full branching-time logic of "eventually"*, while $B^*(\mathsf{X})$ is the *full branching-time logic of "next"*. The existence of a path satisfying an $L(\mathsf{F})$ or an $L(\mathsf{X})$ formula is NP-complete [SC85].
- BTL_2 [RM01] allows path quantifiers where path modalities are given by $FOMLO_2$ formulae, i.e. quantifier-depth 2 formulae in the first-order monadic logic of order. The existence of a path satisfying a $FOMLO_2$ formula is NP-complete [RS00].

For these examples, NP-hardness is easy to prove by reduction from 3SAT. For example, consider an instance $\mathcal{I}$ of the form "$\big(x_1 \vee \overline{x_2} \vee \overline{x_4}\big) \wedge \big(\overline{x_1} \vee \cdots\big) \wedge \cdots$". With $\mathcal{I}$ we associate the following structure that applies to CTL^+, $B^*(\mathsf{F})$, and $B^*(\mathsf{X})$:

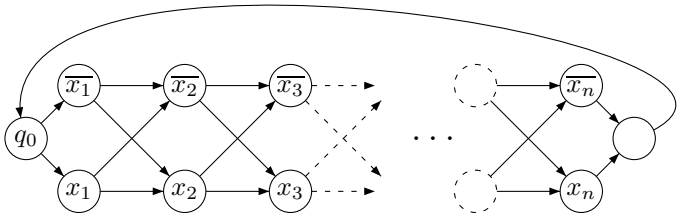


It is easy to see that

$$\mathcal{I} \text{ is satisfiable iff } q_0 \models \mathsf{E}\Big[\big(\mathsf{F}x_1 \vee \mathsf{F}\overline{x_2} \vee \mathsf{F}\overline{x_4}\big) \wedge \big(\mathsf{F}\overline{x_1} \vee \cdots\big) \wedge \cdots\Big] \tag{1}$$

$$\text{iff } q_0 \models \mathsf{E}\Big[\big(\mathsf{X}x_1 \vee \mathsf{XX}\overline{x_2} \vee \mathsf{XXXX}\overline{x_4}\big) \wedge \big(\mathsf{X}\overline{x_1} \vee \cdots\big) \wedge \cdots\Big] \tag{2}$$

For *FCTL* we use a slight variant:

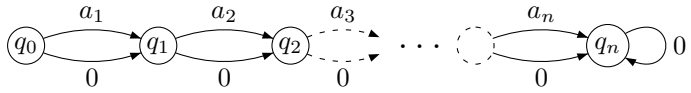


Here $\mathcal{I}$ is satisfiable iff

$$q_0 \models \mathsf{E}\left[\begin{array}{l}\neg(\overset{\infty}{\mathsf{F}} x_1 \wedge \overset{\infty}{\mathsf{F}}\,\overline{x_1}) \wedge \neg(\overset{\infty}{\mathsf{F}} x_2 \wedge \overset{\infty}{\mathsf{F}}\,\overline{x_2}) \wedge \cdots \wedge \neg(\overset{\infty}{\mathsf{F}} x_n \wedge \overset{\infty}{\mathsf{F}}\,\overline{x_n}) \\ \wedge \big(\overset{\infty}{\mathsf{F}} x_1 \vee \overset{\infty}{\mathsf{F}}\,\overline{x_2} \vee \overset{\infty}{\mathsf{F}}\,\overline{x_4}\big) \wedge \big(\overset{\infty}{\mathsf{F}}\,\overline{x_1} \vee \cdots\big) \wedge \cdots\end{array}\right] \tag{3}$$

(1) and (3) also provide proofs for BTL_2 since path modalities in CTL^+ or *FCTL* are directly translated in $FOMLO_2$.

For *TCTL* we reduce from SUBSET-SUM. With an instance $\mathcal{I}$ of the form "can one add numbers taken from $\{a_1, \ldots, a_n\}$ and obtain b?" we associate the following Timed KS:



Obviously

$$\mathcal{I} \text{ is solvable iff } q_0 \models \mathsf{EF}_{=b}\, q_n. \tag{4}$$

2.3 Model checking $B(L)$

Assume L is some linear-time logic, and write $B(L)$ for the associated branching-time logic. Emerson and Lei [EL87] observed that, from an algorithm for the existence of paths satisfying L properties, one easily derives a model checking algorithm for $B(L)$. Furthermore, this only needs a polynomial-time Turing reduction, so that:

Fundamental Observation 2.1 [EL87] *If the existential problem for L belongs to some complexity class $\mathcal{C}$, the model checking problem for $B(L)$ is in $\mathrm{P}^{\mathcal{C}}$.*

Example 2.2. [EL87]. Model checking for *LTL* is PSPACE-complete, thus model checking for CTL^* is in $\mathrm{P}^{\mathrm{PSPACE}}$, that is in PSPACE. Hence, though more expressive than *LTL*, CTL^* is not more expensive for model checking. □

Example 2.3. With path modalities having an NP-complete existential problem, $B^*(\mathsf{F})$, $B^*(\mathsf{X})$, CTL^+, $ECTL^+$ (from [EH86]), *FCTL*, BTL_2 and *TCTL* (over Timed KSs), all have a model checking in P^{NP}, the level called Δ_2^p in the polynomial-time hierarchy.[4] □

[4] For CTL^+ and $B^*(\mathsf{F})$, membership in Δ_2^p was observed as early as [CES86, Theo. 6.2].

2.4 Characterizing the complexity of model checking $B(\boldsymbol{L})$ logics

Concerning the logics mentioned in Example 2.3, the only known lower bounds for their model checking problem were the obvious "NP-hard and coNP-hard" (or even DP-hard). However, all these logics have Θ_2^p-hard model checking (see Remark 5.3 below). Recently, Laroussinie, Markey and Schnoebelen showed Δ_2^p-hardness (hence Δ_2^p-completeness) for *FCTL* and $B^+(\mathsf{F})$ in [LMS01] (hence also for $B^*(\mathsf{F})$, CTL^+, $ECTL^+$, and BTL_2), and for *TCTL* over Timed KSs in [LMS02a].

The Δ_2^p-hardness proof techniques developed in [LMS01,LMS02a] were not able to cope with $B^*(\mathsf{X})$, or with *Timed* $B(\mathsf{F})$ (the fragment of *TCTL* where only the F modality may carry timing subscripts). This raises the question of whether these logics have Δ_2^p-hard model checking, and how to prove that. The Δ_2^p upper-bound is indeed too coarse: in the rest of the paper, we prove that model checking $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete, and model checking *Timed* $B(\mathsf{F})$ is $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-complete.

3 Oracle Boolean circuits and TB(SAT)

3.1 Blocks and trees of blocks

We consider special oracle Boolean circuits called *blocks*. As illustrated in Fig. 1, a block is a circuit B computing an output vector $\boldsymbol{z}$ of k bits from a set $\boldsymbol{y^1}, \ldots, \boldsymbol{y^m}$ of m input vectors, again with k bits each. Inside the block, p internal gates

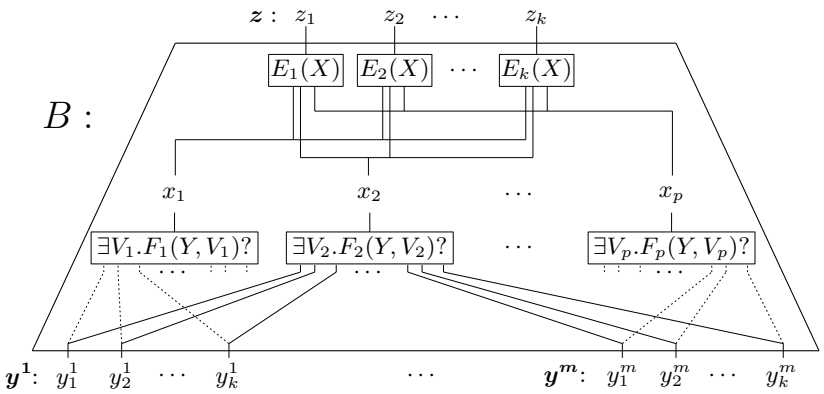


Fig. 1. General form of a "block" oracle circuit

$x_1, \ldots, x_p$ query a SAT oracle: x_i evaluates to 1 iff $F_i(Y, V_i)$ is satisfiable, where F_i is a Boolean formula combining the km input bits $Y = \{y_l^j \mid j = 1, \ldots, m,\ l = 1, \ldots, k\}$ with some additional variables from some set V_i. Finally, the values of the output bits are computed from the x_i's by means of classical Boolean circuits (no oracles): z_i is some $E_i(X)$ where $X = \{x_1, \ldots, x_p\}$. We say m is the *degree*

of the block, k is its *width*, and its size is the usual number of gates *augmented by the sizes of the F_i formulae*.

The obvious algorithm for computing the value of $\boldsymbol{z}$ for some km input bits is a typical instance of $\mathrm{P}_{\parallel}^{\mathrm{NP}}$: the p oracle queries are independent and can be asked in parallel. Building the queries and combining their answers to produce $\boldsymbol{z}$ is a simple polynomial-time computation.

Blocks are used to form more complex circuits: a *tree of blocks* is a circuit T obtained by connecting blocks having a same width k in a tree structure, as illustrated in Fig. 2 (where block B_7 has degree 0). Requiring that all input and

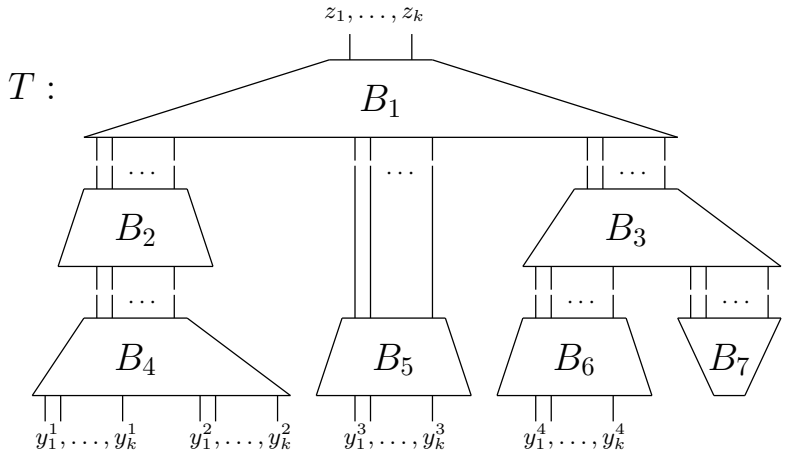


Fig. 2. A "tree of blocks" oracle circuit

output vectors of all blocks have a same width k only simplifies notations (and connections of blocks) but this is no loss of generality, and it would be easy to normalize blocks and trees having heterogeneous vectors.

Every block in a tree has a level defined in the obvious way: in our example, $B_4, \ldots, B_7$ are at level 1, B_2, B_3 at level 2, and B_1, the *root*, at level 3. If the root of some tree is at level d, then the natural way of computing the value of the output $\boldsymbol{z}$ requires d rounds of parallel queries: in our example, the queries inside B_1 can only be formulated after the B_2 queries have been answered, and formulating these require that the B_4 queries have been answered before.

TB(SAT) is the decision problem where one is given a tree of blocks, Boolean values for its input bits, and is asked the value of (one of) the output bits. Compared to the more general problem of evaluating circuits with oracle queries (e.g. the Δ_2^p-complete **DAG**(SAT) problem of [Got95]), we impose the restriction of a tree-like structure, and compared to the more particular problem of evaluating Boolean trees with oracle queries (the Θ_2^p-complete **TREE**(SAT) problem of [Got95]), we allow each node of the tree to transmit a vector of bits to its

parent node. Thus **TB**(SAT) is a restriction of **DAG**(SAT) and a generalization of **TREE**(SAT).

Proposition 3.1. **TB**(SAT) *is* Δ_2^p*-complete.*

Proof (Idea). Membership in Δ_2^p is clear. Hardness for Δ_2^p uses the width of the blocks to circumvent the imposed tree structure: an oracle circuit C with n adaptive SAT queries can be reduced to a "tree" of n stacked blocks B_1 on B_2 on ... on B_n such that the l first output bits of B_{n+1-l} are the answers to the first l queries of C. The resulting tree has size $O(|C|^2)$. □

3.2 Circuits with simple oracle queries

In a block of width k and degree m, we say a query $\exists V.F(Y,V)?$ has *type* 1×M if it has the form $\exists l_1,\ldots,l_m\,\exists V'.F(y^1_{l_1},\ldots,y^m_{l_m},l_1,\ldots,l_m,V')?$, i.e. F only uses one bit from each input vector (but it can be any bit and this is existentially quantified upon). Our formulation quantifies upon indexes $l_1,\ldots,l_m$ in the $1,\ldots,k$ range but such a l_j is a shorthand for e.g. k bits "$l_j{=}1$", ..., "$l_j{=}k$" among which one and only one will be true. These bits are part of V and this is why F depends on $l_1,\ldots,l_m$ (and on V', which is V without the l_js). There is a similar notion of type 2×M, type 3×M, ..., where F only uses 2 (resp. 3, ...) bits from each input vector.

We say that a query has *type* 1×1 if it has the form $\exists j\,\exists l\,\exists V'.F(y^j_l,j,l,V')?$, i.e. F only uses one bit from one input vector (can be any bit from any vector and this is existentially quantified upon). Again, there is a similar notion of type 2×1, type 3×1, ..., where we only use 2 (resp. 3, ...) bits in total.

For a query type τ, we let $\mathbf{TB}(\mathrm{SAT})_\tau$ denote the **TB**(SAT) problem restricted to trees of type τ (i.e. trees where all queries have type τ).

Before we see (in later sections) where such restricted queries appear, we show that they give rise to simpler oracle circuits:

Theorem 3.2. *For any* $n>0$
1. $\mathbf{TB}(\mathrm{SAT})_{n\times 1}$ *is* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$*-complete,*
2. $\mathbf{TB}(\mathrm{SAT})_{n\times \mathrm{M}}$ *is* $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$*-complete.*

We prove the upper bounds in the rest of this section. The lower bounds, Corollaries 4.6 and 5.6, are deduced from hardness results for model checking problems studied in the following sections [5].

[5] More direct hardness proofs are possible, and may have improved the clarity of the paper, but they would not have excused us from proving that our model checking problems are hard. It is easy to construct hardness proofs for circuits by streamlining our hardness proofs for model checking.

3.3 Lowering TB(SAT)$_{1\times M}$ circuits

Assume block B is the parent of some B' inside a type $1\times$M tree T. Fig. 3 illustrates how one can merge B and B' into an equivalent single block B_{new}. Here B' is the leftmost child block of B, so that the input vector $\boldsymbol{y^1}$ of B will play a special role, but the construction could have been applied with any other child.

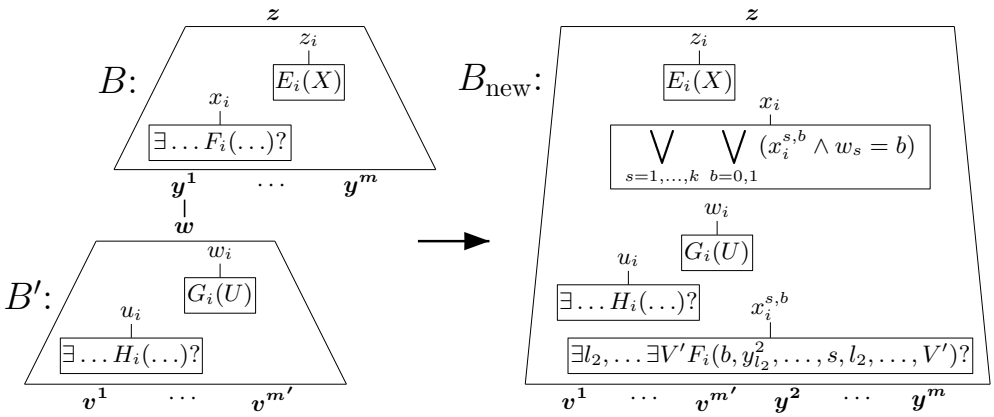


Fig. 3. Merging type $1\times$M blocks

The new block copies the u_i query gates and the G_i circuits from B' without modifying them. $2k$ new query gates $x_i^{s,b}$ are introduced for each x_i in B: $x_i^{s,b}$ is like x_i but it assumes $l_1 = s$ and $y_s^1 = b$ in F_i. The x_i query gates from B are replaced by new (non-query) circuits picking the best $x_i^{s,b}$ for which w_s agrees with the assumed value for y_s^1. The final B_{new} has type $1\times$M and degree $m' + m - 1$. $|B_{\text{new}}|$ is $O(|B'| + 2k|B|)$: B was *expanded* but B' is unchanged.

The purpose of this merge operation is to lower the level of trees: we say a tree is *low* if its root has level at most $\log(1+\text{number of blocks in the tree})$. The tree in Fig. 2 has 7 blocks and root at level 3, so it is (just barely) low.

Lemma 3.3. *There is a logspace reduction that transforms type* $1\times$M *trees of blocks into equivalent low trees.*

Proof. Consider a type $1\times$M tree T. We say a block in T is *bad* if it is at some level $d > 1$ in T and has exactly one child at level $d-1$ (called its *bad child*). For example B_2 is the only bad node in Fig. 2. If T has bad nodes, we pick a bad B of lowest level and merge it with its bad child. We repeat this until T has no bad node: the final tree T_{new} is low since any non-leaf block at level d must have at least two children at level $d-1$ hence at least $2^d - 2$ descendants.

Observe that, when we merge a bad B at level d with its bad child B', the resulting B_{new} has level $d-1$. Also, since we picked B lowest possible, B' was not bad, so B_{new} cannot be bad or have bad descendants. Thus B_{new} will never

be bad again (though it can become a bad child) and will not be expanded a second time. Therefore T_{new} has size $O(k|T|)$ which is $O(|T|^2)$. □

Observe that evaluating a low tree T only requires $O(\log|T|)$ rounds of parallel oracle queries. Therefore Lemma 3.3 provides a reduction from $\mathbf{TB}(\text{SAT})_{1\times\text{M}}$ to $\text{P}^{\text{NP}}_{\|O(\log n)}$, a $\text{P}^{\text{NP}[O(\log^2 n)]}$-complete problem [CS96].

Corollary 3.4. $\mathbf{TB}(\text{SAT})_{1\times\text{M}}$ *is in* $\text{P}^{\text{NP}[O(\log^2 n)]}$.

If now n is any fixed number, the obvious adaptation of the merging technique can lower trees of type $n\times\text{M}$. Here the new block B_{new} uses $(2k)^n$ new query gates but since n is fixed, the transformation is logspace and the resulting T_{new} has size $O(|T|^{n+1})$.

Corollary 3.5. *For any* $n\in\mathbb{N}$, $\mathbf{TB}(\text{SAT})_{n\times\text{M}}$ *is in* $\text{P}^{\text{NP}[O(\log^2 n)]}$.

3.4 Flattening $\mathbf{TB(SAT)_{1\times1}}$ circuits

Lemma 3.6. *For any* $n\in\mathbb{N}$, *there is a logspace reduction that transforms type* $n\times1$ *trees of blocks into equivalent blocks.*

Proof (Sketch). With type 1×1 trees, one can merge all children $B_1,\ldots,B_m$ with their parent B without incurring any combinatorial explosion. A query gate x_i of the form $\exists j\,\exists l\,\exists V'.F_i(y^j_l,j,l,V')?$ will give rise to $2km$ new query gates

$$x_i^{r,s,b} \;:=\; \exists V'.F_i(b,r,s,V')?$$

where r is the assumed value for j, s the assumed value for l and b the assumed value for y^r_s. x_i will now be computed via

$$x_i \;:=\; \bigvee_{r=1,\ldots,m}\;\bigvee_{s=1,\ldots,k}\;\bigvee_{b=0,1}(x_i^{r,s,b}\wedge w^r_s=b).$$

We have $|B_{\text{new}}|=O(|B_1|+\cdots+|B_m|+2km|B|)$ so that a bottom-up repetitive application will transform a type 1×1 tree T into a single block of size $O(|T|^3)$.

For type $n\times1$ trees, the obvious generalization introduces $(2km)^n$ new query gates when merging $B_1,\ldots,B_m$ with their parent B, so that a tree T is flattened into a single block of size $O(|T|^{2n+1})$. □

Lemma 3.6 provides a logspace reduction from $\mathbf{TB}(\text{SAT})_{n\times1}$ to $\text{P}^{\text{NP}}_{\|}$, a $\text{P}^{\text{NP}[O(\log n)]}$-complete problem [BH91].

Corollary 3.7. *For any* $n\in Nat$, $\mathbf{TB}(\text{SAT})_{n\times1}$ *is in* $\text{P}^{\text{NP}[O(\log n)]}$.

4 Model checking $B^*(\mathsf{X})$

In this section we show:

Theorem 4.1. *The model checking problem for $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete.*

We start by introducing BX^*, a fragment of $B^*(\mathsf{X})$ where all occurrences of X are immediately over an atomic proposition, or an existential path quantifier (or an other X). Formally, BX^* is given by the following abstract syntax:

$$\varphi ::= \mathsf{E}f(\mathsf{X}^{n_1}\varphi_1, \ldots, \mathsf{X}^{n_k}\varphi_k) \mid P_1 \mid P_2 \mid \ldots$$

where $f(\ldots)$ is any Boolean formula.

Lemma 4.2. *There exists a logspace transformation of $B^*(\mathsf{X})$ formulae into equivalent BX^* formulae.*

Proof. One just has to bury the X's so that they only apply to atomic propositions or E quantifiers. This relies on

$$\mathsf{X}(\varphi \wedge \psi) \equiv \mathsf{X}\varphi \ \wedge \ \mathsf{X}\psi \qquad\qquad \mathsf{X}(\neg\varphi) \equiv \neg\mathsf{X}\varphi \tag{5}$$

and induces at most a quadratic blowup in size. (One may also have to introduce dummy path quantifiers on top of Boolean connectives, e.g. $P_1 \wedge \varphi$ is transformed into $\mathsf{E}f_\wedge(\mathsf{X}^0 P_1, \mathsf{X}^0\varphi)$.) □

Lemma 4.3. *There exists a logspace transformation from model checking for BX^* into $\mathbf{TB}(\mathrm{SAT})_{1\times\mathrm{M}}$.*

Proof. With a KS S and a BX^* formula φ we associate a tree of blocks where the width k is the number of states in S, and where there is a block B_ψ for every subformula ψ of φ (so that the structure of the tree mimics the structure of φ). The blocks are built in a way that ensures that the ith output bit of B_ψ is true iff q_i, the ith state in S, satisfies ψ. This only needs type $1\times\mathrm{M}$ blocks. Assume ψ is some $\exists f(\mathsf{X}^{n_1}\psi_1, \ldots, \mathsf{X}^{n_m}\psi_m)$ with $n_1 \leq n_2 \leq \ldots \leq n_m$. Then, for $i = 1, \ldots, k$, B_ψ computes whether $q_i \models \psi$ with a query gate x_i defined via

$$x_i \ := \ \exists l_1, \ldots, l_m \Big(f(y^1_{l_1}, \ldots, y^m_{l_m}) \ \wedge \bigwedge_{j=1,\ldots,m} Path(l_{j-1}, n_j - n_{j-1}, l_j) \Big) \ ?$$

where $l_0 = i$, $n_0 = 0$ and $Path(l, n, l')$ (definition omitted) is a Boolean formula stating that S has an n-steps path from q_l to $q_{l'}$. □

Combined with Corollary 3.4, Lemmas 4.2 and 4.3 entail:

Corollary 4.4. *Model checking for $B^*(\mathsf{X})$ is in $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$.*

For Theorem 4.1, we need prove the corresponding lower bound:

Proposition 4.5. *Model checking for $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-hard.*

(See Appendix for a proof.)

Combining with Lemmas 4.2 and 4.3, we also obtain:

Corollary 4.6. $\mathbf{TB}(\mathrm{SAT})_{1\times\mathrm{M}}$ *is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-hard.*

5 Model checking *Timed* $B(\mathsf{F})$

In this section we show:

Theorem 5.1. *The model checking problem for Timed* $B(\mathsf{F})$ *over Timed KSs is* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$*-complete.*

This is obtained through the next two lemmas.

Lemma 5.2. *Model checking Timed* $B(\mathsf{F})$ *over Timed KSs is* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$*-hard.*

Proof. By reduction from PARITY-SAT. Assume we are given a set $\mathcal{I}_0, \ldots, \mathcal{I}_{n-1}$ of SUBSET-SUM instances: we saw in section 2.2 how to associate with these a Kripke structure S and simple *Timed* $B(\mathsf{F})$ formulae $\psi_0, \ldots, \psi_{n-1}$ s.t. for every i, $\mathcal{I}_i$ is solvable iff $S \models \psi_i$. Assume w.l.o.g. that n is some power of 2: $n = 2^d$ and for every tuple $\langle b_1, \ldots, b_k \rangle$ of $k \leq d$ bits define

$$\varphi_{\langle b_1,\ldots,b_k\rangle} \stackrel{\text{def}}{=} \begin{cases} \neg\, \psi_{\sum_{j=1}^{k} b_j 2^{j-1}} & \text{if } k = d, \\ (\varphi_{\langle 0,b_1,\ldots,b_k\rangle} \wedge \varphi_{\langle 1,b_1,\ldots,b_k\rangle}) \vee (\neg\varphi_{\langle 0,b_1,\ldots\rangle} \wedge \neg\varphi_{\langle 1,b_1,\ldots\rangle}) & \text{otherwise.} \end{cases}$$

$S \models \varphi_{\langle b_1,\ldots,b_k\rangle}$ iff there is an even number of solvable $\mathcal{I}_i$s among those whose index i has $b_1, \ldots, b_k$ as last k bits. Therefore the total number of solvable $\mathcal{I}_i$ is even iff $S \models \varphi_{\langle\rangle}$. Since $d = \log n$, $|\varphi_{\langle\rangle}|$ is in $O(n \sum_i |\psi_i|)$ and the reduction is logspace. □

We note that $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$-hardness already occurs with a modal depth 1 formula.

Remark 5.3. Observe that this proof applies to all the logics we mentioned in section 2.2: it only requires that several SAT problems $f_1, \ldots, f_n$ can be reduced to respective formulae $\psi_1, \ldots, \psi_n$ other a same structure S. (This is always possible for logics having a reachability modality like EX or EF). □

Lemma 5.4. *There exists a logspace transformation from model checking for Timed* $B(\mathsf{F})$ *over Timed KSs into* $\mathbf{TB}(\mathrm{SAT})_{1\times 1}$.

Proof (Sketch). We mimic the proof of Lemma 4.3: again we associate a block B_ψ for each subformula and k is the number of states of the Kripke structure S. Assume the edges $e_1, \ldots, e_r$ of S carry weights $d_1, \ldots, d_r$. Then, for ψ of the form $\mathsf{EF}_{=c}\psi'$, block B_ψ will compute whether $q_i \models \psi$ by asking the query

$$x_i \;:=\; \exists l\, \exists n_1, \ldots, n_r \Big(y_l^1 \;\wedge\; c = \sum_{j=1,\ldots,r} n_j r_j \;\wedge\; \mathit{Path}'(i, n_1, \ldots, n_r, l) \Big)\ ?$$

where $\mathit{Path}'(i, n_1, \ldots, n_r, l)$ (definition omitted) is a Boolean formula checking that there exists a path from q_i to q_l that uses exactly n_j times edge e_j for each $j = 1, \ldots, r$ (Euler's circuit theorem makes the check easy). We refer to [LMS02a, Lemma 4.5] for more details (e.g. how are the n_is polynomially bounded?) since here we only want to see that type 1×1 queries are sufficient for *Timed* $B(\mathsf{F})$.

Dealing with $\mathsf{AF}_{=c}\psi$ seems more difficult, especially because of possible 0 weights, but it is actually simpler. We explain how to deal with $\mathsf{EG}_{=c}$, the dual modality: $q_i \models \mathsf{EG}_{=c}\psi$ if there is a path from q_i along which all the states that occur at accumulated weight c satisfy ψ. There are two cases: c never happens along the path (and then $q_i \models \mathsf{EG}_{=c}\bot$) or it does (and then $q_i \models \mathsf{EF}_{=c}\mathsf{EG}_{=0}\psi$). In both cases we need look at the first transition along the path that reaches c or above:

$$q_i \models \mathsf{EG}_{=c}\psi \text{ iff } \left| \begin{array}{l} \text{there is a } c' < c \text{ and an edge } (q, w, q') \text{ in } S \\ \quad \text{s.t.} \left| \begin{array}{l} c' + w \geq c \text{ and } q_i \text{ can reach } q \text{ with accumulated weight } c' \\ \text{and if } c' + w = c \text{ then } q' \models \mathsf{EG}_{=0}\psi \end{array} \right. \\ \text{or} \\ \quad q_i \mathsf{EG}_{\geq c}\bot \end{array} \right.$$

(the last line deals with the case where the path stays forever *below* c). Since model checking the $\mathsf{EG}_{=0}$ and $\mathsf{EG}_{\geq c}$ modalities is polynomial-time (see [LMS02a]), we end up with a SAT query (for the "$\exists c' < c, \mathsf{EF}_{=c'} \ldots$") of type 0×1. □

Corollary 5.5. *Model checking Timed* $B(\mathsf{F})$ *over Timed KSs is in* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$.

Corollary 5.6. $\mathbf{TB}(\mathrm{SAT})_{1\times 1}$ *is* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$*-hard.*

6 Conclusion

We solved the model checking problems for $B^*(\mathsf{X})$ and *Timed* $B(\mathsf{F})$, two temporal logic problems where the precise computational complexity was left open.

For $B^*(\mathsf{X})$, the result is especially interesting because of the fundamental nature of this logic, but also because it provides the first example of a natural problem complete for $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$. Indeed, identifying the right complexity class for this problem was part of the difficulty.

Proving membership in $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$ required introducing a new family of oracle circuits. These circuits are characterized by their tree-vector form, and additional special logical conditions on the way an oracle query may depend on its inputs. The tree-vector form faithfully mimics branching-time model checking, while the special logical conditions originate from the modalities that appear in the path formulae. We expect our results on the evaluation of these circuits will be applied to other branching-time logics.

References

[BH91] S. R. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[CS96] J. Castro and C. Seara. Complexity classes between Θ_k^p and Δ_k^p. *RAIRO Informatique Théorique et Applications*, 30(2):101–121, 1996. A shorter version appeared in Proc. STACS'92, LNCS 577.

[DS02] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

[DW99] M. Dickhöfer and T. Wilke. Timed alternating tree automata: The automata-theoretic solution to the TCTL model checking problem. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 1999.

[EH85] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EL87] E. A. Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[Got95] G. Gottlob. NP trees and Carnap's modal logic. *Journal of the ACM*, 42(2):421–457, 1995.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pages 67–161. Elsevier Science, 1990.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LMS01] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking CTL^+ and $FCTL$ is hard. In *Proc. 4th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2001), Genova, Italy, Apr. 2001*, volume 2030 of *Lecture Notes in Computer Science*, pages 318–331. Springer, 2001.

[LMS02a] F. Laroussinie, N. Markey, and Ph. Schnoebelen. On model checking durational Kripke structures. In *Proc. 5th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2002), Grenoble, France, Apr. 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2002.

[LMS02b] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proc. 17th IEEE Symp. Logic in Computer Science (LICS'2002), Copenhagen, Denmark, July 2002*, pages 383–392. IEEE Comp. Soc. Press, 2002.

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

[PZ83] C. H. Papadimitriou and S. K. Zachos. Two remarks on the power of counting. In *Proc. 6th GI Conf. on Theor. Comp. Sci., Dortmund, FRG, Jan.*

1983, volume 145 of *Lecture Notes in Computer Science*, pages 269–276. Springer, 1983.

[Rab02] A. Rabinovich. Expressive power of temporal logics. In *Proc. 13th Int. Conf. Concurrency Theory (CONCUR'2002), Brno, Czech Republic, Aug. 2002*, volume 2421 of *Lecture Notes in Computer Science*, pages 57–75. Springer, 2002.

[RM01] A. Rabinovich and S. Maoz. An infinite hierarchy of temporal logics over branching time. *Information and Computation*, 171(2):306–332, 2001.

[RS00] A. Rabinovich and Ph. Schnoebelen. BTL_2 and expressive completeness for $ECTL^+$. Research Report LSV-00-8, Lab. Specification and Verification, ENS de Cachan, Cachan, France, October 2000. Submitted for publication. Available at `http://www.lsv.ens-cachan.fr/Publis/`.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sch03] Ph. Schnoebelen. The complexity of temporal logic model checking (invited lecture). In *Advances in Modal Logic, papers from 4th Int. Workshop on Advances in Modal Logic (AiML'2002), Sep.-Oct. 2002, Toulouse, France*. World Scientific, 2003. To appear. A preliminary version is available at `http://www.lsv.ens-cachan.fr/Publis/`.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. Logic in Computer Science (LICS'86), Cambridge, MA, USA, June 1986*, pages 332–344. IEEE Comp. Soc. Press, 1986.

[Wag87] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.

[Wag90] K. W. Wagner. Bounded query classes. *SIAM J. Computing*, 19(5):833–846, 1990.

[Wil87] C. B. Wilson. Relativized NC. *Mathematical Systems Theory*, 20(1):13–29, 1987.

A Proof of Proposition 4.5

A.1 $\mathbf{P}^{\mathbf{NP}[O(\log^2 n)]}$-hardness

Let M be a deterministic polynomial-time oracle TM that makes at most $c \log^2 n$ queries on inputs of size n. We exhibit a logspace reduction from the accepting problem for M^{SAT} to $B^*(\mathsf{X})$ model checking.

For simplification purposes, and without loss of generality, we assume that when running on some input $\boldsymbol{w} \in \{0,1\}^n$ of size n, M makes *exactly* d^2 queries with d given by $2^{d-1} < n \leq 2^d$, and that the outcome of the computation is given by the last oracle query (i.e. M^{SAT} accepts $\boldsymbol{w}$ iff its d^2th query is answered positively).

Given a vector $\boldsymbol{a} = \langle a_1, \ldots, a_{d^2} \rangle$ of d^2 bits, we write $M^{\boldsymbol{a}}$ for the TM that runs like M on inputs of size n, except that its d^2 oracle queries receive the answers $a_1, \ldots, a_{d^2}$ (in that order). We say that $\boldsymbol{a}$ is *exact for* $\boldsymbol{w}$ if $\boldsymbol{a}$ coincide with the answers that M would receive from SAT when run on $\boldsymbol{w}$. Now, since M is deterministic, the computation of M^{SAT} and the computation of $M^{\boldsymbol{a}}$ on $\boldsymbol{w}$

are completely identical when $\boldsymbol{a}$ is exact for $\boldsymbol{w}$. Below we let $r(\boldsymbol{w}) = \langle r_1, \ldots, r_{d^2}\rangle$ denote the only vector that is exact for $\boldsymbol{w}$.

We say that $\boldsymbol{a}$ is *pessimistic for* $\boldsymbol{w}$ if $a_i = 0$ for all i such that the ith query of $M^{\boldsymbol{a}}$ on $\boldsymbol{w}$ is not in SAT. Observe that when the ith query is in SAT, we have no constraint on a_i, so a pessimistic $\boldsymbol{a}$ needs not be exact. Also note that if $\boldsymbol{a}$ is not exact then, after the first "wrong" answer, $M^{\boldsymbol{a}}$ and M^{SAT} will behave differently, and in particular may ask different future queries.

Let $\leq_{lex}$ denote lexicographical ordering. With M we associate the language

$$W_M \stackrel{\text{def}}{=} \{\langle \boldsymbol{w}, \boldsymbol{b}\rangle \mid \boldsymbol{b} \leq_{lex} \boldsymbol{a} \text{ for some } \boldsymbol{a} \text{ pessimistic for } \boldsymbol{w}\}. \tag{6}$$

Lemma A.1. $r(\boldsymbol{w}) = \max_{lex}\{\boldsymbol{b} \mid \langle \boldsymbol{w}, \boldsymbol{b}\rangle \in W_M\}$.

Proof. Obviously $\langle \boldsymbol{w}, r(\boldsymbol{w})\rangle$ is in W_M. Now assume that $r(\boldsymbol{w}) \neq \boldsymbol{a}$ for some pessimistic $\boldsymbol{a}$ and let i be the first position where $r(\boldsymbol{w})$ and $\boldsymbol{a}$ differ: when run on $\boldsymbol{w}$, $M^{\boldsymbol{a}}$ and M^{SAT} ask the same ith query but receive different answers. These must be $r_i = 1$ and $a_i = 0$ since $\boldsymbol{a}$ is pessimistic. Thus $\boldsymbol{a} <_{lex} r(\boldsymbol{w})$. □

Lemma A.2. W_M *is in* NP.

Proof. Given a candidate $\langle \boldsymbol{w}, \boldsymbol{b}\rangle$, a certificate is some $\boldsymbol{a} \geq_{lex} \boldsymbol{b}$ and some satisfying assignments for the queries corresponding to the positive bits in $\boldsymbol{a}$. With this one can check in polynomial-time that indeed $\boldsymbol{a}$ is pessimistic for $\boldsymbol{w}$. □

Thus there is a set $Z_n = \{z_1, \ldots, z_{m_n}\}$ of Boolean variables and a Boolean formula $f_n(w_1, \ldots, w_n, y_1, \ldots, y_{d^2}, Z)$ such that $\langle \boldsymbol{w}, \boldsymbol{b}\rangle \in W_M$ iff $f_n(\boldsymbol{w}, \boldsymbol{b}, Z_n)$ is satisfiable, i.e. iff $\exists Z_n.f_n(\boldsymbol{w}, \boldsymbol{b}, Z_n)$ holds. f_n and m_n depend on n but the proof of Lemma A.2 shows that they can be produced uniformly in logspace from $1^{|\boldsymbol{w}|}$.

Lemma A.1 says that a possible way to decide whether M^{SAT} accepts $\boldsymbol{w}$ is to compute the lexicographically first vector $\boldsymbol{b}$ s.t. $\langle \boldsymbol{w}, \boldsymbol{b}\rangle$ is in W_M and look at its last bit. If an oracle for W_M is available, computing that lexicographically first vector can be done by d^2 rounds of binary search. Our reduction encodes a similar search but it uses d rounds where, at each round, d bits are computed.

With $\boldsymbol{w}$ we associate the KS S_n depicted in Fig. 4. S_n has a clique of 2^d states labeled $q_0, q_1, \ldots, q_{2^d-1}$. For any k with $0 \leq k < 2^d$, the state q_k encodes a packet of d bits that we denote $bits(k)$. In S_n it is possible to leave the clique, move to the s state and pick a valuation for the Boolean variables $z_1, \ldots, z_{m_n}$ that appear in f_n.

We now give $B^*(\mathsf{X})$ formulae $(\varphi_i)_{1\leq i\leq d}$ stating that $bits(k)$ is the ith packet of d bits in $r(\boldsymbol{w})$. Formally we define inductively:

$$\psi_i \stackrel{\text{def}}{=} \mathsf{X}^{d+1}s \wedge \bigwedge_{p<2^d}\big(q_p \Leftrightarrow \mathsf{X}^i q_p\big) \wedge \bigwedge_{1\leq j<i} \mathsf{X}^j\mathsf{E}\varphi_j \wedge \theta_{f_n(\boldsymbol{w})}, \tag{7}$$

$$\varphi_i \stackrel{\text{def}}{=} \psi_i \wedge \mathsf{A}\Big[\big(\bigvee_{p<p'<2^d}(q_p \wedge \mathsf{X}q_{p'})\big) \Rightarrow \mathsf{X}\neg\mathsf{E}\psi_i\Big], \tag{8}$$

where $\theta_{f_n(\boldsymbol{w})}$ will be defined in the proof of Lemma A.3.

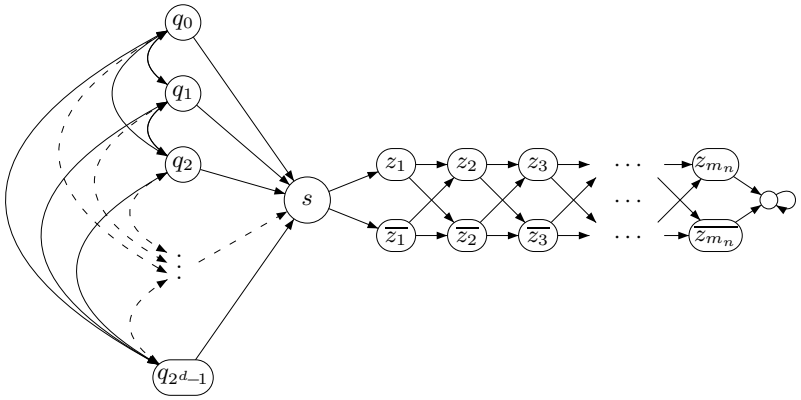


Fig. 4. Kripke structure for $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-hardness of $B^*(\mathsf{X})$

Lemma A.3. *Let* $r(\boldsymbol{w}) = \langle r_1, \ldots, r_{d^2}\rangle$. *Then for all* $k < 2^d$ *and* $1 \leq i \leq d$

$$q_k \models \mathsf{E}\psi_i \;\; \textit{iff} \;\; \mathit{bits}(k) \leq_{lex} \langle r_{(i-1)d+1}, r_{(i-1)d+2}, \ldots, r_{i.d}\rangle, \tag{9}$$

$$q_k \models \mathsf{E}\varphi_i \;\; \textit{iff} \;\; \mathit{bits}(k) \;\; = \;\; \langle r_{(i-1)d+1}, r_{(i-1)d+2}, \ldots, r_{i.d}\rangle. \tag{10}$$

Proof. By induction on i.

We start with the ($\Rightarrow$) direction of (9). Assume $q_k \models \mathsf{E}\psi_i$, i.e. $\pi \models \psi_i$ for some path π. Then $\pi \models \mathsf{X}^{d+1}s$ and π has the form $q_k, q_{k_1}, \ldots, q_{k_d}, s, \pm_1 z_1, \pm_2 z_2, \ldots$ where every $\pm_j z_j$ is the literal z_j or $\overline{z_j}$.

We see π as a valuation for the free variables in $f_n(\boldsymbol{w})$: $k_1, \ldots, k_d$ provide values for $y_1, \ldots, y_{d^2}$ and the tail of π provides values for the z_js. $\theta_{f_n(\boldsymbol{w})}$ is a $B^*(\mathsf{X})$ path formula stating that these values satisfy $f_n(\boldsymbol{w})$. It is obtained from f_n by the same direct translation we used in (2): a z_j is simply replaced with $\mathsf{X}^{d+1+j}z_j$ and a w_i by the actual value (from $\boldsymbol{w}$). Replacing the y_js is only slightly more involved: if j is $(l-1)d + l'$ for some $1 \leq l, l' \leq d$, then y_j receives the l'th bit of k_l, so we replace y_j by $\mathsf{X}^l\big(\bigvee\{q_p \mid \mathit{bits}(p)_{l'} = 1\}\big)$. Finally, $\theta_{f_n(\boldsymbol{w})}$ ensures that $\langle \boldsymbol{w}, \mathit{bits}(k_1)\cdots\mathit{bits}(k_d)\rangle$ is in W_M as witnessed by the $\pm_j z_j$s picked by π. Hence $\mathit{bits}(k_1)\cdots\mathit{bits}(k_d) \leq_{lex} r(\boldsymbol{w})$.

The states $q_{k_1}, q_{k_2}, \ldots, q_{k_{i-1}}$ satisfy (respectively) $\mathsf{E}\varphi_1, \mathsf{E}\varphi_2, \ldots, \mathsf{E}\varphi_{i-1}$ so that, by ind. hyp., $\mathit{bits}(k_1), \ldots, \mathit{bits}(k_{i-1})$ are the first $(i-1)d$ bits of $r(\boldsymbol{w})$. Thus $\mathit{bits}(k_i) \leq_{lex} \langle r_{(i-1)d+1}, r_{(i-1)d+2}, \ldots, r_{id}\rangle$. We conclude by observing that $k = k_i$ since $\pi \models \bigwedge_p (q_p \Leftrightarrow \mathsf{X}^i q_p)$.

After these explanations, the ($\Leftarrow$) direction is easy to see. Assume $\mathit{bits}(k) \leq \langle r_{(i-1)d+1}, r_{(i-1)d+2}, \ldots, r_{id}\rangle$ and consider a $\boldsymbol{b}$ obtained by putting the $(i-1)d$ bits of $r(\boldsymbol{w})$ in front of $\mathit{bits}(k)$ and appending $(d-i)d$ zeros at the end. $\boldsymbol{b} \leq_{lex} r(\boldsymbol{w})$ and $f(\boldsymbol{w}, \boldsymbol{b}, Z_n)$ is satisfiable. From this we build a π that proves $q_k \models \mathsf{E}\psi_i$.

Finally (10) is clear since $q_k \models \mathsf{E}\varphi_i$ simply states that k is the largest number s.t. $q_k \models \mathsf{E}\psi_i$. □

Corollary A.4. M^{SAT} *accepts* $\boldsymbol{w}$ *iff* $q_0 \models \mathsf{EX}\big(\varphi_d \wedge \bigvee\{q_{2p+1} \mid 0 \leq p < 2^{d-1}\}\big)$.

Proof. M^{SAT} accepts $\boldsymbol{w}$ iff the last bit of $r(\boldsymbol{w})$ is 1. □

It remains to evaluate the size of φ_d: ψ_i contains $\varphi_1, \ldots, \varphi_{i-1}$ and the rest can be polynomially bounded, while φ_i contains two occurrences of ψ_i and some polynomially bounded rest. If we write $p(n)$ for the polynomial bound, an easy induction shows that

$$|\psi_i| \leq \frac{3^i - 1}{2} p(n) \qquad\qquad |\varphi_i| \leq 3^i p(n) \tag{11}$$

Thus $|\varphi_d| \leq 3^d p(n)$ which is $O(n \times p(n))$ since $d = \log n$, and Corollary A.4 provides the logspace reduction we announced. Hence

Corollary A.5. *Model checking $B^*(\mathsf{X})$ is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-hard.*