The Tractability of Model-Checking for LTL: The Good, the Bad, and the Ugly Fragments*

Michael Bauland[1], Martin Mundhenk[2], Thomas Schneider[3], Henning Schnoor[4], Ilka Schnoor[5], and Heribert Vollmer[5]

[1] Knipp GmbH, Martin-Schmeißer-Weg 9, 44227 Dortmund, Germany
—Michael.BaulandATknipp.de—
[2] Informatik, Friedrich-Schiller-Universität, 07737 Jena, Germany
—martin.mundhenkATuni-jena.de—
[3] Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK
—schneiderATcs.man.ac.uk—
[4] Theoret. Informatik, Christian-Albrechts-Universität, 24098 Kiel, Germany
—schnoorATti.informatik.uni-kiel.de—
[5] Theoret. Informatik, Universität Lübeck, 23538 Lübeck, Germany
—schnoorATtcs.uni-luebeck.de—
[6] Theoret. Informatik, Universität Hannover, Appelstr. 4, 30167 Hannover, Germany
—vollmerATthi.uni-hannover.de—

Abstract. In a seminal paper from 1985, Sistla and Clarke showed that the model-checking problem for Linear Temporal Logic (LTL) is either NP-complete or PSPACE-complete, depending on the set of temporal operators used. If, in contrast, the set of propositional operators is restricted, the complexity may decrease. This paper systematically studies the model-checking problem for LTL formulae over restricted sets of propositional and temporal operators. For almost all combinations of temporal and propositional operators, we determine whether the model-checking problem is tractable (in P) or intractable (NP-hard). We then focus on the tractable cases, showing that they all are NL-complete or even logspace solvable. This leads to a surprising gap in complexity between tractable and intractable cases. It is worth noting that our analysis covers an infinite set of problems, since there are infinitely many sets of propositional operators.

* Supported in part by DFG VO 630/6-1 and the Postdoc Programme of the German Academic Exchange Service (DAAD).

1 Introduction

Linear Temporal Logic (LTL) has been proposed by Pnueli [**?**] as a formalism to specify properties of parallel programs and concurrent systems, as well as to reason about their behaviour. Since then, it has been widely used for these purposes. Recent developments require reasoning tasks—such as deciding satisfiability, validity, or model checking—to be performed automatically. Therefore, decidability and computational complexity of the corresponding decision problems are of great interest.

The earliest and fundamental source of complexity results for the satisfiability problem (SAT) and the model-checking problem (MC) of LTL is certainly Sistla and Clarke's paper [**?**]. They have established PSPACE-completeness of SAT and MC for LTL with the temporal operators F (eventually), G (invariantly), X (next-time), U (until), and S (since). They have also shown that these problems are NP-complete for certain restrictions of the set of temporal operators. This work was continued by Demri and Schnoebelen, and Markey [**?**, **?**], who established the complexity of satisfiability and model checking for fragments of LTL. Other fragments of LTL have been examined, although not for MC, by Kučera and Strejček, and Muscholl and Walukiewicz [**?**, **?**]. The results of Sistla and Clarke, Markey, and Muscholl and Walukiewicz imply that SAT and MC for LTL and a multitude of its fragments are intractable. In fact, only Demri and Schnoebelen do exhibit tractable fragments.

The fragments they consider are obtained by restricting the set of temporal operators and the use of negations. What they do not consider are arbitrary fragments of temporal *and* Boolean operators. For propositional logic, a complete analysis has been achieved by Lewis [**?**]. He divides all infinitely many sets of Boolean operators into those with tractable (polynomial-time solvable) and intractable (NP-complete) SAT problems. A similar systematic classification has been obtained by Bauland et al. in [**?**] for LTL. They divide fragments of LTL—determined by arbitrary combinations of temporal and Boolean operators—into those with polynomial-time solvable, NP-complete, and PSPACE-complete SAT problems.

This paper continues the work on the MC problem for LTL with the temporal *future* operators F, G, X, U, and R (release). Similarly

as in [?], the considered fragments are arbitrary combinations of temporal and Boolean operators. The five listed temporal operators are of course not the only operators that have been considered for LTL. For instance, their past counterparts are common [?, ?], and it is also possible to consider operators defined by LTL formulae, such as the ternary operator $\mathsf{O}(\alpha, \beta, \gamma) = \mathsf{F}\alpha \vee (\beta \mathsf{U} \gamma)$, or based on automata [?]. Further possibilities to enhance expressiveness of LTL are discussed in [?]. Since we believe that the combinatorial explosion caused by considering *all* such temporal operators cannot be handled, we are restricting this study to the above listed five future operators. We have good reasons to believe that they are the most relevant operators:

- Future operators are used, and indeed suffice, for formulating many standard properties in verification, such as various versions of fairness and liveliness, see for instance [?].
- The operators normally used in "modern" model checking are the five listed above, see for instance [?] or the manual of the model checker Spin [?]. Vardi in his historical account [?] introduces basic future linear temporal logic in terms of the operators X and U, and states that other operators can be defined in terms of these—F and G are explicitly mentioned.
- Properties relevant for the security of cryptographic protocols are expressed in alternating temporal logic (ATL) [?, ?, ?], and future operators are sufficient.

It should be noted that the infinitely many Boolean operators are not problematic, due to the techniques applied to systematise all infinitely many relevant sets of Boolean operators. Unfortunately, this technique cannot be expected to be transferable to temporal operators because their definitions are well beyond propositional logic.

We will separate the MC problem for almost all LTL fragments into tractable (*i.e.*, polynomial-time solvable) and intractable (*i.e.*, NP-hard) cases. This extends work of Sistla and Clarke, Demri and Schnoebelen, and Markey [?, ?, ?], but in contrast to their results, we will exhibit many tractable fragments and exactly determine their computational complexity. Surprisingly, we will see that tractable cases for model checking are even very easy—that is, NL-complete or even L-solvable. There is only one set of Boolean operators, consisting

of the binary *xor*-operator, that we will have to leave open. This constellation has already proved difficult to handle in [**?**, **?**], the latter being a paper where SAT for basic modal logics has been classified in a similar way.

While the borderline between tractable and intractable fragments in [**?**, **?**] is quite easily recognisable (SAT for fragments containing the Boolean function $f(x, y) = x \wedge \overline{y}$ is intractable, for all others it is tractable), our results for MC will exhibit a rather diffuse borderline. This will become visible in the following overview and is addressed in the Conclusion. Our most surprising intractability result is the NP-hardness of the fragment that only allows the temporal operator U (resp. only its dual R) and no propositional operator at all. Our most surprising tractability result is the NL-completeness of MC for the fragment that only allows the temporal operators F, G, and the binary *or*-operator. Taking into account that MC for the fragment with only F plus *and* is already NP-hard (which is a consequence from [**?**]), we would have expected the same lower bound for the "dual" fragment with only G plus *or*, but in fact we show that even the fragment with F and G and *or* is tractable. In the presence of the X-operator, the expected duality occurs: The fragment with F, X plus *and* and the one with G, X plus *or* are both NP-hard.

Table 1 gives an overview of our results. The top row refers to the sets of Boolean operators given in Definition 2.3. These seven sets of Boolean operators are all relevant cases, which is due to Post's fundamental paper [**?**] and Lemma 2.2. Entries in bold-face type denote completeness for the given complexity class under logspace reductions. (All reductions in this paper are logspace many-one reductions $\leq_m^{\log}$, cf. [**?**].) The entry L stands for logspace solvability. All other entries denote hardness results. Superscripts refer to the source of the corresponding result as explained in the legend.

This paper is organised as follows. Section 2 contains all necessary definitions and notation. In Section 3, we show NP-hardness of all intractable cases, followed by Section 4 with the NL-completeness of almost all remaining cases. We conclude in Section 5.

prop. operators / temp. operators	I	N	E	V	M	L	BF
X	$\mathbf{NL}^7$	$\mathbf{NL}^7$	$\mathbf{NL}^9$	$\mathbf{NL}^8$	NP^2	$\mathbf{NL}^{11}$	NP^S
G	$\mathbf{NL}^7$	$\mathbf{NL}^7$	$\mathbf{NL}^9$	$\mathbf{NL}^{10}$	NP^2		NP^S
F	$\mathbf{NL}^7$	$\mathbf{NL}^7$	NP^4	$\mathbf{NL}^8$	NP^2		NP^S
FG	$\mathbf{NL}^7$	$\mathbf{NL}^7$	NP^c	$\mathbf{NL}^{10}$	NP^c		NP^S
FX	$\mathbf{NL}^7$	$\mathbf{NL}^7$	NP^c	$\mathbf{NL}^8$	NP^c		PS^T
GX	$\mathbf{NL}^7$	$\mathbf{NL}^7$	$\mathbf{NL}^9$	NP^5	PS^3		PS^T
FGX	$\mathbf{NL}^7$	$\mathbf{NL}^7$	NP^c	NP^c	PS^1		PS^T
all other combinations (*i.e.*, with U or R)	NP^6	NP^c	NP^c	NP^c	PS^3	NP^c	PS^T

Legend.

(PS stands for PSPACE.)

1	Theorem 3.2 1
2	Theorem 3.2 2
3	Theorem 3.2 3
4	Corollary 3.3
5	Theorem 3.4
6	Theorem 3.5
7	Theorem 4.2
8	Theorem 4.3 1
9	Theorem 4.3 2
10	Theorem 4.4
11	Theorem 4.5
S	Theorem 2.1 1
T	Theorem 2.1 2
c	conclusion from surrounding results

Table 1. An overview of complexity results for the model-checking problem

2 Preliminaries

A *Boolean function* is a function $f : \{0,1\}^n \to \{0,1\}$. We can identify an n-ary propositional function symbol c with the n-ary Boolean function f defined by: $f(a_1, \ldots, a_n) = 1$ if and only if the formula $c(x_1, \ldots, x_n)$ becomes true when assigning a_i to x_i for all $1 \leq i \leq n$. An *operator* is either a function or a function symbol, which becomes clear from the context. Additionally to propositional operators we use the unary temporal operators X (next-time), F (eventually), G (invariantly) and the binary temporal operators U (until), and R (release).

Let VAR be a countably infinite set of propositional variables. For any finite sets B of Boolean operators and T of temporal operators, a *temporal B-formula over T* is a formula φ that is built from variables, propositional operators from B, and temporal operators from T. More formally, a temporal B-formula over T is either a propositional variable or of the form $f(\varphi_1, \ldots, \varphi_n)$ or $g(\varphi_1, \ldots, \varphi_m)$, where φ_i are temporal B-formulae over T, f is an n-ary proposi-

tional operator from B and g is an m-ary temporal operator from T. In [**?**], complexity results for formulae using the temporal operators F, G, X (unary), and U, R (binary) were presented. We extend these results to temporal B-formulae over subsets of those temporal operators. The set of variables appearing in φ is denoted by $\mathrm{VAR}(\varphi)$. If $T = \{\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}\}$ we call φ a *temporal B-formula*, and if $T = \emptyset$ we call φ a *propositional B-formula* or simply a *B-formula*. The set of all temporal B-formulae over T is denoted with $\mathrm{L}(T, B)$.

Temporal formulae are interpreted over paths of subsequent states, which intuitively can be seen as different points of time, with propositional assignments. A possibly infinite set of such paths is called a *model* for a formula. We consider models that are represented by Kripke structures. A *Kripke structure* is a triple $K = (W, R, \eta)$, where W is a finite set of states, $R \subseteq W \times W$ is a total binary relation (meaning that, for each $a \in W$, there is some $b \in W$ such that aRb)[7], and $\eta : W \to 2^{\mathrm{VAR}_K}$ for a finite set $\mathrm{VAR}_K \subseteq \mathrm{VAR}$ of variables. A *path* π in K is an infinite sequence denoted as $(\pi_0, \pi_1, \dots)$, where, for all $i \geq 0$, $\pi_i \in W$ and $\pi_i R \pi_{i+1}$.

For a Kripke structure $K = (W, R, \eta)$, a path π in K, and a temporal $\{\wedge, \neg\}$-formula over $\{\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{U}, \mathsf{R}\}$ with variables from VAR_K, we define what it means that π *satisfies* φ *in* π_i ($\pi, i \vDash \varphi$): let φ_1 and φ_2 be temporal $\{\wedge, \neg\}$-formulae over $\{\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{U}, \mathsf{R}\}$ and let $x \in \mathrm{VAR}_K$ be a variable.

[7] In the strict sense, Kripke structures can have arbitrary binary relations. However, when referring to Kripke structures, we always assume their relations to be total.

$\pi, i \vDash 1$ and $\pi, i \nvDash 0$

$\pi, i \vDash x$ iff $x \in \eta(p_i)$

$\pi, i \vDash \varphi_1 \wedge \varphi_2$ iff $\pi, i \vDash \varphi_1$ and $\pi, i \vDash \varphi_2$

$\pi, i \vDash \neg\varphi_1$ iff $\pi, i \nvDash \varphi_1$

$\pi, i \vDash \mathsf{F}\varphi_1$ iff there is a $j \geq i$ such that $\pi, j \vDash \varphi_1$

$\pi, i \vDash \mathsf{G}\varphi_1$ iff for all $j \geq i$ holds $\pi, j \vDash \varphi_1$

$\pi, i \vDash \mathsf{X}\varphi_1$ iff $\pi, i+1 \vDash \varphi_1$

$\pi, i \vDash \varphi_1 \mathsf{U} \varphi_2$ iff there is an $\ell \geq i$ such that $\pi, \ell \vDash \varphi_2$, and for every $i \leq j < \ell$, $\pi, j \vDash \varphi_1$

$\pi, i \vDash \varphi_1 \mathsf{R} \varphi_2$ iff either for every $\ell \geq i$ holds $\pi, \ell \vDash \varphi_2$ or there is an $\ell \geq i$ such that $\pi, \ell \vDash \varphi_1$ and for every $i \leq j \leq \ell$ holds $\pi, j \vDash \varphi_2$

Since every Boolean operator can be composed from $\wedge$ and $\neg$, the above definition generalises to temporal B-formulae for arbitrary sets B of Boolean operators. Two formulae φ and ψ are said to be *equivalent*, denoted $\varphi \equiv \psi$, if, for every path π and every $i \geq 0$, the following are equivalent: $\pi, i \vDash \varphi$ and $\pi, i \vDash \psi$.

A path π is said to *satisfy* a given formula φ if $\pi, 0 \vDash \varphi$. A formula φ is said to be *satisfiable in a state $a \in W$ of a structure* $K = (W, R, \eta)$ if there is a path π starting at a such that p satisfies φ. These two perspectives on satisfiability give rise to two notions of the satisfiability problem. However, it is standard to consider the path-based version:

Problem:	$\mathrm{SAT}(T, B)$
Input:	$\langle\varphi\rangle$, where $\varphi \in \mathrm{L}(T, B)$ is a formula
Question:	Is there a path π such that $\pi, 0 \vDash \varphi$?

This paper examines the model-checking problems $\mathrm{MC}(T, B)$ for finite sets B of Boolean functions and sets T of temporal operators.

Problem:	$\mathrm{MC}(T, B)$
Input:	$\langle\varphi, K, a\rangle$, where $\varphi \in \mathrm{L}(T, B)$ is a formula, $K = (W, R, \eta)$ is a Kripke structure[8], and $a \in W$ is a state

[8] Remember $|W| < \infty$.

Question: Is there a path π in K such that $\pi_0 = a$ and $\pi, 0 \vDash \varphi$?

In both these problems, we assume that the input formula is represented as a string, not as a circuit or DAG, which would allow for more succinctness and would certainly affect some of the complexity bounds proven. We assume familiarity with standard notions of complexity theory (cf. [**?**]). Since the input of the model-checking problem consists of a formula φ, a structure K and a state a, the complexity of $\mathrm{MC}(T, B)$ is measured in the sum of the sizes of all three inputs. We consider simple representations of structures which contain an entry for each state and one for each edge (pair of states), as opposed to more condensed representations which could also encode infinite structures. Furthermore, this version of the model-checking problem is also called *existential*. The *universal* one, in contrast, asks whether *all* paths starting at a in K satisfy φ [**?**]. In the following, we will omit this distinction because we will only consider the existential version.

Sistla and Clarke [**?**] have established the computational complexity of the model-checking problem for temporal $\{\wedge, \vee, \neg\}$-formulae over some sets of temporal operators.[9]

Theorem 2.1 ([?]).

(1) $\mathrm{MC}(\{\mathsf{F}\}, \{\wedge, \vee, \neg\})$ *is* NP*-complete.*
(2) $\mathrm{MC}(\{\mathsf{F}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$, $\mathrm{MC}(\{\mathsf{U}\}, \{\wedge, \vee, \neg\})$, $\mathrm{MC}(\{\mathsf{U}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$, $\mathrm{MC}(\{\mathsf{R}\}, \{\wedge, \vee, \neg\})$,[10] *and* $\mathrm{MC}(\{\mathsf{R}, \mathsf{X}\}, \{\wedge, \vee, \neg\})$[10] *are* PSPACE*-complete.*

Since there are infinitely many finite sets of Boolean functions, we introduce some algebraic tools to classify the complexity of the infinitely many arising model checking problems. We denote with id_k^n the n-ary projection to the k-th variable, where $1 \leq k \leq n$, *i.e.*, $\mathrm{id}_k^n(x_1, \ldots, x_n) = x_k$, and with c_a^n the n-ary constant function defined by $c_a^n(x_1, \ldots, x_n) = a$. For $c_1^1(x)$ and $c_0^1(x)$ we simply write 1 and 0. A set C of Boolean functions is called a *clone* if it is closed

[9] In [**?**] the model-checking problem is called *determination of truth in an R-structure.*
[10] The release-operator R is not used in [**?**], but the result follows immediately since $\neg(\varphi_1 \mathsf{U} \varphi_2) \equiv \neg\varphi_1 \mathsf{R} \neg\varphi_2$ (i.e. the duality of U and R).

under superposition, which means C contains all projections and C is closed under arbitrary composition [**?**]. For a set B of Boolean functions we denote with $[B]$ the smallest clone containing B and call B a *base* for $[B]$. In [**?**] Post classified the lattice of all clones and found a finite base for each clone.

The definitions of all clones as well as the full inclusion graph can be found, for example, in [**?**]. The following lemma implies that only clones with both constants $0, 1$ are relevant for the model-checking problem; hence we will only define those clones. Note, however, that our results will carry over to all clones.

Lemma 2.2. *Let B be a finite set of Boolean functions and T be a set of temporal operators. Then* $\mathrm{MC}(T, B \cup \{0,1\}) \equiv_m^{\log} \mathrm{MC}(T, B)$.

Proof. $\mathrm{MC}(T, B) \leq_m^{\log} \mathrm{MC}(T, B \cup \{0,1\})$ is trivial. For $\mathrm{MC}(T, B \cup \{0,1\}) \leq_m^{\log} \mathrm{MC}(T, B)$ let $\langle \varphi, K, a \rangle$ be an instance of $\mathrm{MC}(T, B \cup \{0,1\})$ for a Kripke structure $K = (W, R, \eta)$ and let $\bot$ and $\top$ be two fresh variables. We define a new Kripke structure $K' = (W, R, \eta')$ with $\mathrm{VAR}_{K'} = \mathrm{VAR}_K \cup \{\bot, \top\}$ and $\eta'(\alpha) = \eta(\alpha) \cup \{\top\}$, and we define φ' to be a copy of φ where every appearance of 0 is replaced by $\bot$ and every appearance of 1 by $\top$. It holds that $\langle \varphi', K', a \rangle$ is an instance of $\mathrm{MC}(T, B)$ and that $\langle \varphi, K, a \rangle \in \mathrm{MC}(T, B \cup \{0,1\})$ if and only if $\langle \varphi', K', a \rangle \in \mathrm{MC}(T, B)$. ❏

Because of Lemma 2.2 it is sufficient to look only at the clones with constants, which are introduced in Definition 2.3. Their bases and inclusion structure are given in Figure 1.

clone	base
BF	$\{\wedge, \neg\}$
M	$\{\vee, \wedge, 0, 1\}$
L	$\{\oplus, 1\}$
V	$\{\vee, 1, 0\}$
E	$\{\wedge, 1, 0\}$
N	$\{\neg, 1, 0\}$
I	$\{0, 1\}$

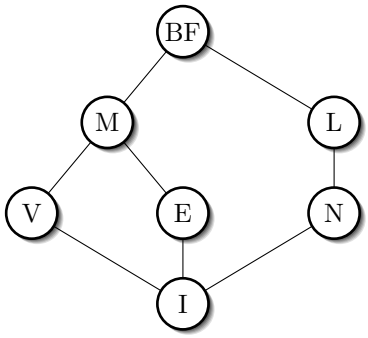

Fig. 1. Clones with constants

Definition 2.3. *(1)* BF *is the set of all Boolean functions.*

(2) M *is the set of all monotone functions, that is, the set of all functions f where $a_1 \leq b_1, \ldots, a_n \leq b_n$ implies $f(a_1, \ldots, a_n) \leq f(b_1, \ldots, b_n)$.*

(3) L *is the set of all linear functions, that is, the set of all functions f that satisfy $f(x_1, \ldots, x_n) = c_0 \oplus (c_1 \wedge x_1) \oplus \cdots \oplus (c_n \wedge x_n)$, for constants c_i. Here, $\oplus$ denotes the binary exclusive or given by $a \oplus b = 1$ iff $a \neq b$.*

(4) V *is the set of all functions f where $f(x_1, \ldots, x_n) = c_0 \vee (c_1 \wedge x_1) \vee \cdots \vee (c_n \wedge x_n)$, for constants c_i.*

(5) E *is the set of all functions f where $f(x_1, \ldots, x_n) = c_0 \wedge (c_1 \vee x_1) \wedge \cdots \wedge (c_n \vee x_n)$, for constants c_i.*

(6) N *is the set of all functions that depend on at most one variable.*

(7) I *is the set of all projections and constants.*

Here, f denotes arbitrary Boolean functions of arity n, for arbitrary n.

The constants $c_1, \ldots, c_n$ in (3)–(5) are used as "switches" for the arguments $x_1, \ldots, x_n$ to achieve that the clone L (V, E) also contains the compositions of n-ary projections and the m-ary exclusive *or* (disjunction, conjunction), for $m < n$.

There is a strong connection between propositional formulae and Post's lattice. If we interpret propositional formulae as Boolean functions, it is obvious that $[B]$ includes exactly those functions that can be represented by B-formulae. This connection has been used various times to classify the complexity of problems related to propositional formulae. For example, Lewis presented a dichotomy for the satisfiability problem for propositional B-formulae: it is NP-complete if $x \wedge \overline{y} \in [B]$, and solvable in P otherwise [**?**]. Furthermore, Post's lattice has been applied to the equivalence problem [**?**], to counting [**?**] and finding minimal [**?**] solutions, and to learnability [**?**] for Boolean formulae. The technique has been used in non-classical logic as well: Bauland et al. achieved a trichotomy in the context of modal logic, which says that the satisfiability problem for modal formulae is, depending on the allowed propositional connectives, PSPACE-complete, coNP-complete, or solvable in P [**?**]. For the inference problem for propositional circumscription, Nordh presented another trichotomy theorem [**?**].

An important tool in restricting the length of the resulting formula in many of our reductions is the following lemma.

Lemma 2.4. *Let $B \subseteq \{\wedge, \vee, \neg\}$, and let C be a finite set of Boolean functions such that $B \subseteq [C]$. Then $\mathrm{MC}(T, B) \leq_m^{\log} \mathrm{MC}(T, C)$ for every set T of temporal operators.*

Proof. Let $D = C \cup \{0, 1\}$. From Lemma 4 in [**?**] we directly conclude: Let f be one of the functions *or*, *and*, and *not* such that $f \in [D]$. Let k be the arity of f. Then there is a D-formula $\varphi(x_1, \ldots, x_k)$ representing f, such that every variable occurs only once in φ. For instance, if $C = \{\oplus, \vee\}$[11], then there are several ways to represent the function $f(x, y) = x \wedge y$ using operators in $[D]$, among them the representation $\big((x \oplus y) \oplus (x \oplus y)\big) \oplus (x \vee y)$ and $1 \oplus \big((1 \oplus x) \oplus (1 \oplus y)\big)$. The former contains each variable three times, which causes an exponential blowup if $\wedge$-operators are nested. The latter representation avoids this blowup. Because of Lemma 4 in [**?**], such a representation always exists.

We can now successively replace every *or*-, *and*-, and *not*- subformula of any given $\mathrm{L}(T, B)$-formula with the respective D-formula φ. We therefore have $\mathrm{MC}(T, B) \leq_m^{\log} \mathrm{MC}(T, C \cup \{0, 1\})$. From Lemma 2.2 it follows that $\mathrm{MC}(T, C \cup \{0, 1\}) \leq_m^{\log} \mathrm{MC}(T, C)$. ❑

It is essential for our proof of this Lemma that $B \subseteq \{\wedge, \vee, \neg\}$. *E.g.*, it is open whether $\mathrm{MC}(T, \{\oplus\}) \leq_m^{\log} \mathrm{MC}(T, \{\wedge, \vee, \neg\})$. The reason is that it causes an exponential blowup to transform an $\mathrm{L}(T, \{\oplus\})$-formula into an equivalent $\mathrm{L}(T, \{\wedge, \vee, \neg\})$-formula.

Nevertheless, the PSPACE upper bound for model checking proven by Sistla and Clarke [**?**] holds for all sets of Boolean functions—i.e. for all sets $T \subseteq \{\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{U}, \mathsf{R}\}$ of temporal operators and all finite sets B of Boolean functions, $\mathrm{MC}(T, B) \in$ PSPACE. The reason is that every $\mathrm{L}(T, B)$-formula can efficiently be transformed into an equivalent $\mathrm{L}(T, \{\wedge, \vee, \neg\})$-formula that is encoded by a data structure that contains every subformula at most once—for example one can take directed acyclic graphs (dags). The PSPACE algorithm by Sistla and Clarke keeps its polynomial space upper bound if it gets formulae represented by dags as input.

[11] Here, $\oplus$ denotes the binary exclusive or given by $a \oplus b = 1$ iff $a \neq b$.

3 The bad fragments: intractability results

Sistla and Clarke [**?**] and Markey [**?**] have considered the complexity of model-checking for temporal $\{\wedge, \vee, \neg\}$-formulae restricted to atomic negation and propositional negation, respectively. We define a temporal B-formula with *propositional negation* to be a temporal B-formula where additional negations are allowed, but only in such a way that no temporal operator appears in the scope of a negation sign. In the case that negation is an element of B, a temporal B-formula with propositional negation is simply a temporal B-formula. In [**?**], *atomic negation* is considered, which restricts the use of negation even further—negation is only allowed directly for variables. We will now show that propositional negation does not make any difference for the complexity of the model checking problem. Since this obviously implies that atomic negation inherits the same complexity behaviour, we will only speak about propositional negation in the following. The proof of the following lemma is similar to that of Lemma 2.2.

Lemma 3.1. *Let T be a set of temporal operators, and B a finite set of Boolean functions. We use $\mathrm{MC}^+(T, B)$ to denote the model-checking problem $\mathrm{MC}(T, B)$ extended to B-formulae with propositional negation. Then $\mathrm{MC}^+(T, B) \equiv_m^{\log} \mathrm{MC}(T, B)$.*

Proof. The reduction $\mathrm{MC}(T, B) \leq_m^{\log} \mathrm{MC}^+(T, B)$ is trivial. For $\mathrm{MC}^+(T, B) \leq_m^{\log} \mathrm{MC}(T, B)$, assume that negation is not an element of B, otherwise there is nothing to prove. Let $\langle \varphi, K, a \rangle$ be an instance of $\mathrm{MC}^+(T, B)$, where $K = (W, R, \eta)$. Let $x_1, \ldots, x_m$ be the variables that appear in φ, and for each formula of the kind $\neg\psi(x_1, \ldots, x_n)$ appearing in φ, let $y_{\neg\psi}$ be a new variable. Note that since only propositional negation is allowed in φ, in these cases ψ is purely propositional.

We obtain $K' = (W, R, \eta')$ from K by extending η to the variables $y_{\neg\psi}$ in such a way that $y_{\neg\psi}$ is *true* in a state if and only if $\psi(x_1, \ldots, x_n)$ is *false*. Finally, to obtain φ' from φ we replace every appearance of $\neg\psi(x_1, \ldots, x_n)$ with $y_{\neg\psi}$. This can be done in logarithmic space because it primarily involves counting of parentheses. Now, φ' is a temporal B-formula. By the construction it is straightforward to see that $\langle \varphi, K, a \rangle \in \mathrm{MC}^+(T, B)$ iff $\langle \varphi', K', a \rangle \in \mathrm{MC}(T, B)$. ❑

Using Lemma 2.4 in addition, we can generalise the above mentioned hardness results from [**?**, **?**] for temporal monotone formulae to obtain the following intractability results for model-checking.

Theorem 3.2. *Let M_+ be a finite set of Boolean functions such that* $\mathrm{M} \subseteq [M_+]$. *Then*

(1) $\mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, M_+)$ *is* PSPACE*-hard.*
(2) $\mathrm{MC}(\{\mathsf{F}\}, M_+)$, $\mathrm{MC}(\{\mathsf{G}\}, M_+)$, *and* $\mathrm{MC}(\{\mathsf{X}\}, M_+)$ *are* NP*-hard.*
(3) $\mathrm{MC}(\{\mathsf{U}\}, M_+)$, $\mathrm{MC}(\{\mathsf{R}\}, M_+)$, *and* $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, M_+)$ *are* PSPACE*-hard.*

In Theorem 3.5 in [**?**] it is shown that $\mathrm{MC}(\{\mathsf{F}\}, \{\wedge, \vee, \neg\})$ is NP-hard. In fact, Sistla and Clarke give a reduction from 3SAT to $\mathrm{MC}(\{\mathsf{F}\}, \{\wedge\})$. The result for arbitrary bases B generating a clone above E follows from Lemma 2.4.

Corollary 3.3. *Let E_+ be a finite set of Boolean functions such that* $\mathrm{E} \subseteq [E_+]$. *Then* $\mathrm{MC}(\{\mathsf{F}\}, E_+)$ *is* NP*-hard.*

The model-checking problem for temporal $\{\mathsf{G}, \mathsf{X}\}$-$\{\wedge, \vee\}$-formulae is PSPACE-complete (Theorem 3.23 due to [**?**]). The Boolean operators $\{\wedge, \vee\}$ are a basis of M, the class of monotone Boolean formulae. What happens for fragments of M? In Theorem 4.3 we will show that $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, \mathrm{E})$ is NL-complete, *i.e.*, the model-checking problem for temporal $\{\wedge\}$-formulae over $\{\mathsf{G}, \mathsf{X}\}$ is very simple. We can prove that switching from $\wedge$ to $\vee$ makes the problem intractable. As notation, we use $\mathrm{LIT}(\varphi)$ to denote the literals obtained from variables that appear in φ.

Theorem 3.4. *Let V_+ be a finite set of Boolean functions such that* $\mathrm{V} \subseteq [V_+]$. *Then* $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, V_+)$ *is* NP*-hard.*

Proof. It suffices to give a reduction from 3SAT to $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, \{\vee\})$ (due to Lemma 2.4). A formula ψ in 3CNF is mapped to an instance $\langle \psi', K(\psi), q_1 \rangle$ of $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, \{\vee\})$ as follows. Let $\psi = C_1 \wedge \ldots \wedge C_m$ consist of m clauses, and $n = |\,\mathrm{VAR}(\psi)|$ variables. The Kripke structure $K(\psi)$ has states $Q = \{q_1, \ldots, q_m\}$ containing one state for every clause, a sequence of states $P = \{l^j \mid l \in \mathrm{LIT}(\psi), 0 \leq j \leq m-1\}$ for every literal, and a final sink state z. That is, the set of

states is $Q \cup P \cup \{z\}$. The variables of $K(\psi)$ are $b_1, \ldots, b_m, c$. Variable b_a is assigned *true* in a state l_i^0 iff literal l_i is contained in clause C_a. In all other states, every b_i is *false*. Variable c is assigned *true* in all states in $P \cup \{z\}$.

The relation between the states is $E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$ as follows. It starts with the path $q_1, \ldots, q_m$: $E_1 = \{(q_i, q_{i+1}) \mid i = 1, 2, \ldots, m-1\}$. q_m has an edge to x_1^0 and an edge to $\overline{x}_1^0$: $E_2 = \{(q_m, x_1^0), (q_m, \overline{x}_1^0)\}$. Each l_i^0 is the starting point of a path $l_i^0, l_i^1, \ldots, l_i^{m-1}$: $E_3 = \{(l_i^j, l_i^{j+1}) \mid l_i \in \mathrm{LIT}(\psi), j = 0, 1, \ldots, m-2\}$. Each endpoint of these paths has both the literals with the next index resp. the final sink state as neighbours: $E_4 = \{(l_i^{m-1}, l_{i+1}^0) \mid i = 1, 2, \ldots, n-1, l_i \in \mathrm{LIT}(\psi)\} \cup \{(x_n^{m-1}, z), (\overline{x}_n^{m-1}, z)\}$. The final sink state z has an edge to z itself, $E_5 = \{(z, z)\}$.

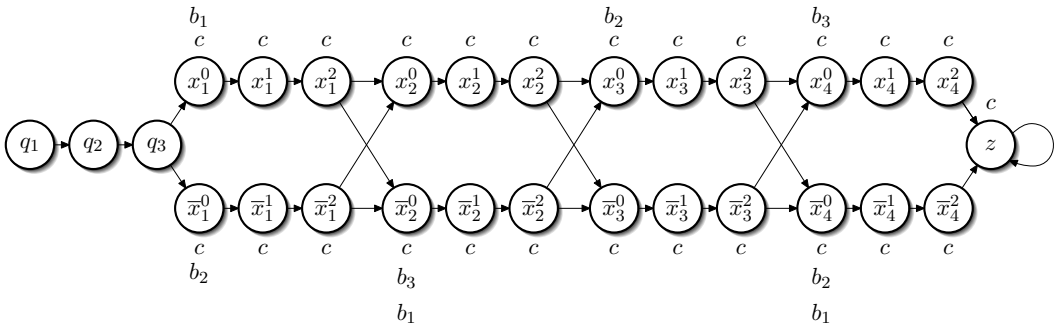

Fig. 2. The Kripke structure $K(\psi_0)$ for $\psi_0 = (x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4)$.

Figure 2 shows an example for a formula ψ_0 and the Kripke structure $K(\psi_0)$. Notice that every path in such a Kripke structure $K(\psi)$ corresponds to an assignment to the variables in ψ. A path corresponds to a satisfying assignment iff for every b_i the path contains a state that b_i is assigned to. We are now going to construct a formula ψ' to express this property. If we were allowed to use the $\wedge$ in ψ', this would be easy. But, the formula ψ' consists only of operators G, X, $\vee$, and of variables $b_1, \ldots, b_m, c$. In order to define ψ', we use formulae φ_i and ψ_i' defined as follows. For $i = 1, 2, \ldots, m$ define

$$\varphi_i = \bigvee_{k=1,2,\ldots,n} \mathsf{X}^{k \cdot m - (i-1)} b_i \ .$$

Intuitively, φ_i says that b_i is satisfied in a state in distance d, where $d \equiv m-(i-1) \pmod m$. The state q_j is the only state in Q where φ_j can hold. Every path p in $K(\psi)$ has the form $p = (q_1, q_2, \ldots, q_m, l_1^0, \ldots, l_n^{m-1}, z, z, \ldots)$. Every state except for z appears at most once in p. For the sake of simplicity, for a path $\pi = (q_1, q_2, \ldots)$ in $K(\psi)$ we use the notation $\pi, q_i \vDash \alpha$ for $\pi, i-1 \vDash \alpha$ (for $i = 1, 2, \ldots, m$), and $\pi, l_i^j \vDash \alpha$ for $\pi, m + (i-1) \cdot m + j \vDash \alpha$.

Claim 1. For every path π in $K(\psi)$ and $1 \leq i, j \leq m$ holds: If $\pi, q_i \vDash \varphi_j$, then $i = j$.

Proof of Claim 1. Assume $\pi, q_i \vDash \varphi_j$, where $1 \leq i, j \leq m$. By the definition of φ_j, it follows that $\pi, (j-1) + k \cdot m - (i-1) \vDash b_j$ for some k with $1 \leq k \leq n$. Consider any path p in $K(\psi)$. After the initial part $(p_0, \ldots, p_{m-1}) = (q_1, \ldots, q_m)$ of p follows a sequence $(p_m, \ldots, p_{m\cdot(n+1)})$ of $n \cdot m$ states, where $p_i = l_{\lfloor i/m \rfloor}^{i \bmod m}$ (for $i = m, \ldots, m\cdot(n+1)$). Therefore, $p_{(j-1)+k\cdot m-(i-1)} = l_r^{(j+k\cdot m-i) \bmod m} = l_r^{(j-i) \bmod m}$ for some r (that does not matter here). But $p, l_r^w \vDash b_j$ implies $w = 0$, by definition of $K(\psi)$, and therefore $(j-i) \bmod m = 0$. Since $1 \leq i, j \leq m$, it follows that $j = i$. ∎

The formulae ψ'_i are defined inductively for $i = m+1, \ldots, 2, 1$ as follows (as before, we can use $\vee$ in our construction):

$$\psi'_{m+1} = c \qquad \text{and} \qquad \psi'_i = \mathsf{G}\left(\varphi_i \vee \mathsf{G}\psi'_{i+1}\right) \text{ (for } m \geq i \geq 1) \,.$$

Finally, $\psi' = \psi'_1$.

It is clear that the reduction function $\psi \mapsto \langle \psi', K(\psi), q_1 \rangle$ can be computed in logarithmic space. It remains to prove the correctness of the reduction. Using Claim 1, we make the following observation.

Claim 2. For every path $\pi = (q_1, q_2, \ldots)$ in $K(\psi)$ and $i = 1, 2, \ldots, m$ holds:

$\pi, q_i \vDash \psi'_i$ if and only if for $j = i, i+1, \ldots, m$ holds $\pi, q_j \vDash \varphi_j$.

Proof of Claim 2. The direction from right to left is straightforward. To prove the other direction, we use induction.

As base case we consider $i = m$. Assume $\pi, q_m \vDash \mathsf{G}(\varphi_m \vee \mathsf{G}c)$. By construction of $K(\psi)$ holds $\pi, q_m \nvDash c$, and therefore $\pi, q_m \vDash \varphi_m$ holds.

For the inductive step, assume $\pi, q_i \vDash \mathsf{G}(\varphi_i \vee \mathsf{G}\psi'_{i+1})$. Claim 1 proves $\pi, q_i \nvDash \varphi_j$ for $j \neq i$, and with $\pi, q_i \nvDash c$ we obtain $\pi, q_i \nvDash \mathsf{G}\psi'_{i+1}$. This implies $\pi, q_i \vDash \varphi_i$ and $\pi, q_{i+1} \vDash \psi'_{i+1}$. By the inductive hypothesis, the claim follows. ∎

For a path π in $K(\psi)$, let $\mathcal{A}_\pi$ be the corresponding assignment for ψ. It is clear that $\pi, q_i \vDash \varphi_i$ if and only if $\mathcal{A}_\pi$ satisfies clause C_i of ψ. Using Claim 2, it follows that $\pi, q_1 \vDash \psi'$ if and only if $\mathcal{A}_\pi$ satisfies all clauses of ψ, *i.e.*, $\mathcal{A}_\pi$ satisfies ψ. Using the one-to-one correspondence between paths in $K(\psi)$ and assignments to the variables of ψ we get $\psi \in 3\text{SAT}$ if and only if $\langle \psi', K(\psi), q_1 \rangle \in \text{MC}(\{\mathsf{G}, \mathsf{X}\}, \{\vee\})$. ❑

From [**?**] it follows that $\text{MC}(\{\mathsf{G}, \mathsf{X}\}, \text{V})$ is in PSPACE. It remains open whether $\text{MC}(\{\mathsf{G}, \mathsf{X}\}, \text{V})$ or $\text{MC}(\{\mathsf{G}, \mathsf{X}\}, \text{M})$ have an upper bound *below* PSPACE.

Next, we consider formulae with the until-operator or the release-operator.
Markey [**?**] showed that $\text{MC}(\{\mathsf{U}\}, \text{M})$ is PSPACE-complete. We show that using the until-operator or the release-operator makes model-checking intractable even in the absence of any Boolean operators.

Theorem 3.5. *Let B be a finite set of Boolean functions. Then* $\text{MC}(\{\mathsf{U}\}, B)$ *and* $\text{MC}(\{\mathsf{R}\}, B)$ *are* NP*-hard.*

Proof. We first prove that $\text{MC}(\{\mathsf{U}\}, B)$ is NP-hard by giving a reduction from 3SAT to $\text{MC}(\{\mathsf{U}\}, \emptyset)$. This means, that we do not need any Boolean operators in the temporal formula over $\{\mathsf{U}\}$ to which a 3SAT instance is mapped. Let $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a 3CNF formula consisting of m clauses and n variables. The structure $K(\psi)$ has states $\{q_1, \ldots, q_m\} \cup \text{LIT}(\psi) \cup \{s\}$, with initial state q_1. The assignment for state q_i is $\{a_1, \ldots, a_i\}$ (for $i = 1, 2, \ldots, m$), and for state l_i it is $\{a_1, \ldots, a_m\} \cup \{b_j \mid \text{Literal } l_i \in \text{LIT}(\psi) \text{ appears in clause } C_j\}$. In state s, no variable is assigned true. The relation between the states is as follows. Each q_i $(i = 1, 2, \ldots, m-1)$ has an edge to q_{i+1}, q_m has edges to x_1 and to $\overline{x}_1$, each x_i and each $\overline{x}_i$ $(i = 1, 2, \ldots, n-1)$ has edges to both x_{i+1} and $\overline{x}_{i+1}$, and x_n and $\overline{x}_n$ have an edge to s. s has an edge to s only. Figure 3 gives an example. The following facts are easy to verify for any path p in $K(\psi)$. For the sake of simplicity, we use for a path $\pi = (q_1, q_2, \ldots)$ in $K(\psi)$ and $1 \leq i \leq m$ the notation $\pi, q_i \vDash \alpha$ for $\pi, i-1 \vDash \alpha$.

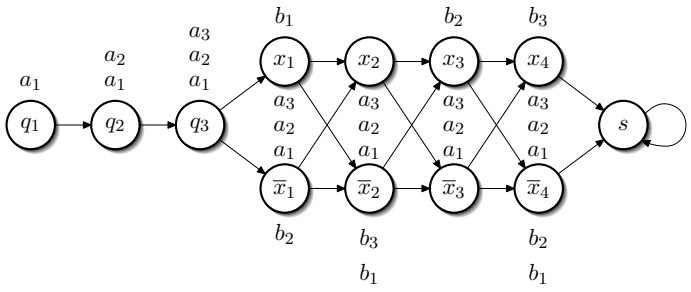

Fig. 3. Structure $K(\psi)$ for $\psi = (x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_4)$

Fact 1 For $1 \leq j < i \leq m$ holds: $\pi, q_j \not\models a_i \mathsf{U} b_i$.
Fact 2 For $1 \leq i \leq m$ holds: $\exists t : \pi, t \models a_i \mathsf{U} b_i$ iff $\pi, q_i \models a_i \mathsf{U} b_i$.

The formulae $\varphi_0, \varphi_1, \ldots$ are defined inductively as follows.

$$\varphi_0 \;=\; 1 \qquad \text{and} \qquad \varphi_{i+1} \;=\; \varphi_i \mathsf{U} \, (a_{i+1} \mathsf{U} b_{i+1}) \, .$$

The reduction from 3SAT to $\mathrm{MC}(\{\mathsf{U}\}, \emptyset)$ is the mapping $\psi \mapsto (\varphi_m, K(\psi), q_1)$, where ψ is a 3CNF-formula with m clauses. This reduction can evidently be performed in logarithmic space. To prove its correctness, we use the following claim.

Claim 3. Let $K(\psi)$ be constructed from a formula ψ with m clauses, and let π be a path in $K(\psi)$. For $j = 1, 2, \ldots, m$, it holds that $\pi, q_1 \models \varphi_j$ if and only if $\pi, q_1 \models \varphi_{j-1}$ and $\pi, q_j \models a_j \mathsf{U} b_j$.

Proof of Claim 3. We prove the claim by induction. The base case $j = 1$ is straightforward: $\pi, q_1 \models 1 \, \mathsf{U} (a_1 \mathsf{U} b_1)$ is equivalent to $\exists t : \pi, t \models (a_1 \mathsf{U} b_1)$ which by Fact 2 is equivalent to $\pi, q_1 \models a_1 \mathsf{U} b_1$. The inductive step is split into two cases. First, assume $\pi, q_1 \models \varphi_{j+1}$. Since $\varphi_{j+1} = \varphi_j \mathsf{U} (a_{j+1} \mathsf{U} b_{j+1})$, it follows that $\exists t : \pi, t \models a_{j+1} \mathsf{U} b_{j+1}$. Using Fact 2, we conclude $\pi, q_{j+1} \models a_{j+1} \mathsf{U} b_{j+1}$. By Fact 1, $\pi, q_1 \not\models a_{j+1} \mathsf{U} b_{j+1}$. By the initial assumption, this leads to $\pi, q_1 \models \varphi_j$. Second, assume $\pi, q_1 \models \varphi_j$ and $\pi, q_{j+1} \models a_{j+1} \mathsf{U} b_{j+1}$. Using the induction hypothesis, we obtain $\pi, q_i \models a_i \mathsf{U} b_i$ for $i = 1, 2, \ldots, j+1$. By the construction of φ_{j+1} we immediately get $\pi, q_1 \models \varphi_{j+1}$. ∎

We have a one-to-one correspondence between paths in $K(\psi)$ and assignments to variables of ψ. For a path π we will denote the corresponding assignment by $\mathcal{A}_\pi$. Using Claim 3, it is easy to see that the following properties are equivalent.

1. $\mathcal{A}_\pi$ is a satisfying assignment for ψ.
2. Path π in $K(\psi)$ contains for every $i = 1, 2, \ldots, m$ a state with assignment b_i.
3. $\pi, q_i \vDash a_i \mathsf{U} b_i$ for $i = 1, 2, \ldots, m$.
4. $\pi, q_1 \vDash \varphi_m$.

This concludes the proof that $\psi \in$ 3SAT if and only if $\langle \varphi_m, K(\psi), q_1 \rangle \in \mathrm{MC}(\{\mathsf{U}\}, \emptyset)$.

Essentially the same proof idea also works for the release-operator, showing that $\mathrm{MC}(\{\mathsf{R}\}, B)$ is NP-hard, too. For a given 3CNF-formula ψ with m clauses, construct a structure $K(\psi)$ as above in the case for U. The formulae $\varphi'_0, \varphi'_1, \ldots, \varphi'_{m+1}$ are defined inductively as follows.

$$\varphi'_{m+1} \;=\; 1 \qquad \text{and} \qquad \varphi'_i \;=\; \varphi_{i+1} \mathsf{R} \, (b_i \mathsf{R} a_i) \;\text{ for } 1 \leq i \leq m.$$

The reduction from 3SAT to $\mathrm{MC}(\{\mathsf{R}\}, \emptyset)$ is the mapping $\psi \mapsto (\varphi'_0, K(\psi), q_1)$, where ψ is a 3CNF-formula with m clauses. This reduction can evidently be performed in logarithmic space. Its correctness can be shown in the same way as for U. To give some intuition, consider the formula ψ and the structure $K(\psi)$ from Figure 3. The formula φ'_1 is $((1\mathsf{R}(b_3\mathsf{R}a_3))\mathsf{R}(b_2\mathsf{R}a_2))\mathsf{R}(b_1\mathsf{R}a_1)$. Notice that $b_3\mathsf{R}a_3$ is satisfied on all paths beginning at q_3 that correspond to assignments that satisfy the third clause of ψ. The same holds for $1\mathsf{R}(b_3\mathsf{R}a_3)$. On paths that start in q_2, neither $b_3\mathsf{R}a_3$ nor $1\mathsf{R}(b_3\mathsf{R}a_3)$ can be satisfied. Since $b_2\mathsf{R}a_2$ is satisfied on paths beginning in q_2 that correspond to assignments satisfying the second clause, $(1\mathsf{R}(b_3\mathsf{R}a_3))\mathsf{R}(b_2\mathsf{R}a_2)$ is satisfied on paths from q_2 that satisfy the second clause and the third one. Similar arguments show that φ'_1 can be satisfied only on paths from q_1 that satisfy all clauses of ψ. ❑

An upper bound better than PSPACE for the intractable cases with the until-operator or the release-operator remains open. We will now show that one canonical way to prove an NP upper bound fails, in showing that these problems do not have the "short path property", which claims that a path in the structure that fulfills the formula has length polynomial in the length of the structure and the formula. Hence, it will most likely be nontrivial to obtain a better upper bound.

We will now sketch such families of structures and formulae using an inductive definition. Let $G_1, G_2, \ldots$ be the family of graphs presented in Figures 4 and 5. Notice that G_i is inserted into G_{i+1} using

the obvious lead-in and lead-out arrows. The truth assignments for

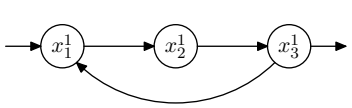

Fig. 4. The graph G_1

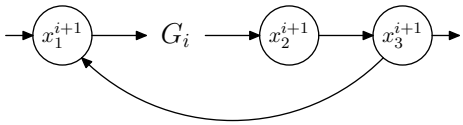

Fig. 5. The graph G_{i+1}

these graphs are as follows:

$$G_1 : \begin{array}{c|l} x_1^1 & b_1 \\ \hline x_2^1 & a_1 \\ \hline x_3^1 & a_1, c_1 \end{array} \qquad G_{i+1} : \begin{array}{c|l} x_1^{i+1} & \bigwedge_{j=1}^{i+1} a_j \\ \hline x_2^{i+1} & a_{i+1} \\ \hline x_3^{i+1} & \bigwedge_{j=1}^{i+1} a_j, c_{i+1} \\ \hline x \in G_i & \text{truth assignment from } G_i, b_{i+1} \end{array}$$

Now the formulae are defined as follows:

$$\varphi_1 = (a_1 \mathsf{U} b_1) \mathsf{U} c_1, \qquad \text{and} \qquad \varphi_{i+1} = ((a_{i+1} \mathsf{U} \varphi_i) \mathsf{U} b_{i+1}) \mathsf{U} c_{i+1}.$$

The rough idea behind the construction is as follows: To satisfy the formula φ_1 in G_1, the path has to repeat the circle once. In the inductive construction, this leads to an exponential number of repetitions. We will now define "canonical paths" in these graphs. These are meant to be the shortest paths in the graph G_i possible that satisfy the formula φ_i. These paths are as follows:

$$\begin{aligned} P_1 &= x_1^1, x_2^1, x_3^1, x_1^1, x_2^1, x_3^1, \\ P_{i+1} &= x_1^{i+1}, P_i, x_2^{i+1}, x_3^{i+1}, x_1^{i+1}, P_i, x_2^{i+1}, x_3^{i+1}. \end{aligned}$$

We claim that the path P_i is a minimal path satisfying φ_i in G_i with the property that it goes back to the out-going edge. The last condition is needed for the recursion, a minimal path satisfying φ_i in G_i can be insignificantly shorter.

We first show that $(a_i \downarrow)^* P_i \vDash \varphi_i$. Here $(a_i \downarrow)^*$ denotes a sequence of states where a_i holds, and no other variable with index of at most i holds. For $i = 1$, this is easy to verify. Therefore, let the claim hold for i. Now, it holds that

$$\varphi_{i+1} = \overbrace{(\underbrace{(a_{i+1} \mathsf{U} \varphi_i)}_{\psi'_{i+1}} \mathsf{U} b_{i+1})}^{\psi''_{i+1}} \mathsf{U} c_{i+1}.$$

For $i+1$, the path P_{i+1} and the truth assignments are as follows (here, $a_j \downarrow$ for some index j denotes the conjunction $a_1 \wedge \cdots \wedge a_j$). In the first row of the table, we list the states, in the second row we list the truth assignments to the variables. The following two rows follow from the above and the induction hypothesis. (We do not claim that the list of subformulae holding in the states presented here is complete, we just highlight those which suffice to show that φ_{i+1} holds.)

$(a_{i+1}\downarrow)^*$	x_1^{i+1}	P_i	x_2^{i+1}	x_3^{i+1}	x_1^{i+1}	P_i	x_2^{i+1}	x_3^{i+1}
$a_{i+1}\downarrow$	$a_{i+1}\downarrow$	b_{i+1}	a_{i+1}	$a_{i+1}\downarrow$ c_{i+1}	$a_{i+1}\downarrow$	b_{i+1}	a_{i+1}	$a_{i+1}\downarrow$ c_{i+1}
φ_i	φ_i			φ_i	φ_i			
ψ'_{i+1}	ψ'_{i+1}		ψ'_{i+1}	ψ'_{i+1}	ψ'_{i+1}			
ψ''_{i+1}	ψ''_{i+1}	ψ''_{i+1}	ψ''_{i+1}	ψ''_{i+1}	ψ''_{i+1}	ψ''_{i+1}		
φ_{i+1}	φ_{i+1}	φ_{i+1}	φ_{i+1}	φ_{i+1}				φ_{i+1}

We claim that P_i is the shortest path satisfying φ_i in G_i. This is clear for $i = 1$. Now suppose that there is a "shortcut" in G_{i+1}. If the shortcut does not use the back-edge (x_3^{i+1}, x_1^{i+1}), at position x_2^{i+1} (this is only visited once, therefore this position is unique), $\mathsf{F} b_{i+1}$ does not hold, and thus ψ''_{i+1} does not hold in x_2^{i+1}. This is a contradiction. Therefore, we pass G_i at least twice.

We show that both times that we pass G_i, we need to follow the full path P_i. We first look at the first passing through G_i. In the path through G_{i+1}, the formula φ_i must hold at the first state from G_i. Since a_i, b_i, and c_i all do not hold in x_2^{i+1}, and no variables with a smaller index, and path fulfilling a smaller $\varphi_j, j < i$, is interrupted here. Hence it follows that φ_i must be satisfied by the path through G_i, by induction hypothesis, this means that G_i has to be passed using the path P_i.

When we pass G_{i+1} for the second time, φ_i again must hold at the first state from G_i and by the same argument as above, this means that G_i again must be passed using the path P_i.

We disregarded the fact that in the very last occurrence of P_j for $j \leq i+1$ the path does not have to pass the whole path P_j but this makes the relevant part of the path only linearly shorter ($2 \cdot (i+1)$ states less).

The length of P_i is obviously exponential in the size of the graph plus the size of the formula.

4 The good fragments: tractability results

This subsection is concerned with fragments of LTL that have a tractable model-checking problem. We will provide a complete analysis for these fragments by proving that model checking for all of them is NL-complete or even solvable in logarithmic space. This exhibits a surprisingly large gap in complexity between easy and hard fragments.

The following lemma establishes NL-hardness for all tractable fragments.

Lemma 4.1. *Let B be a finite set of Boolean functions. Then* $\mathrm{MC}(\{\mathsf{F}\}, B)$, $\mathrm{MC}(\{\mathsf{G}\}, B)$, *and* $\mathrm{MC}(\{\mathsf{X}\}, B)$ *are* NL*-hard.*

Proof. First consider $\mathrm{MC}(\{\mathsf{F}\}, B)$. We reduce the accessibility problem for digraphs, GAP, to $\mathrm{MC}(\{\mathsf{F}\}, \emptyset)$. The reduction is via the following logspace computable function. Given an instance $\langle G, a, b\rangle$ of GAP, where $G = (V, E)$ is a digraph and $a, b \in V$, map it to the instance $\langle \mathsf{F}y, K(G), a\rangle$ of $\mathrm{MC}(\{\mathsf{F}\}, \emptyset)$ with $K(G) = (V, E^+, \eta)$, where E^+ denotes the reflexive closure of E, and η is given by $\eta(b) = \{y\}$ and $\eta(v) = \emptyset$, for all $v \in V - \{b\}$. It is immediately clear that there is a path from a to b in G if and only if there is a path π in $K(G)$ starting from a such that $\pi, 0 \vDash \mathsf{F}y$.

For $\mathrm{MC}(\{\mathsf{X}\}, B)$, we use an analogous reduction from GAP to $\mathrm{MC}(\{\mathsf{X}\}, \emptyset)$. Given an instance $\langle G, a, b\rangle$ of GAP, where $G = (V, E)$, transform it into the instance $\langle \mathsf{X}^{|V|}y, K(G), a\rangle$ of $\mathrm{MC}(\{\mathsf{X}\}, \emptyset)$ with the Kripke structure $K(G)$ from above. Now it is clear that there is a path from a to b in G if and only if there is a path of length $|V|$ from a to b

in the reflexive structure $K(G)$, if and only if there is a path π in $K(G)$ starting from a such that $\pi, 0 \vDash \mathsf{X}^{|V|} y$.

Now consider $\mathrm{MC}(\{\mathsf{G}\}, B)$. We reduce the following problem to $\mathrm{MC}(\{\mathsf{G}\}, \emptyset)$. Given a directed graph $G = (V, E)$ and a vertex $a \in V$, is there an infinite path in G starting at a? It is folklore that this is an NL-hard problem. Given an instance $\langle G, a \rangle$ of this problem, transform it into the instance $\langle \mathsf{G}y, K'(G), a \rangle$ of $\mathrm{MC}(\{\mathsf{G}\}, \emptyset)$, where $K'(G) = (V', E', \eta)$. Here $V' = V \dot{\cup} \{\tilde{v} \mid v \in V,\ v \text{ has no successor in } V\}$, $E' = E \cup \{(v, \tilde{v}),\ (\tilde{v}, \tilde{v}) \mid \tilde{v} \in V'\}$, $\eta(v) = y$ for all $v \in V$, and $\eta(\tilde{v}) = \emptyset$, for all $\tilde{v} \in V'$. It is immediately clear that there is an infinite path in G starting at a if and only if there is a path π in $K'(G)$ starting from a such that $\pi, 0 \vDash \mathsf{G}y$. ❑

It now remains to establish upper complexity bounds. Let C be one of the clones N, E, V, and L, and let B be a finite set of Boolean functions such that $[B] \subseteq C$. Whenever we want to establish NL-membership for some problem $\mathrm{MC}(\cdot, B)$, it will suffice to assume that formulae are given over one of the bases $\{\neg, 0, 1\}$, $\{\wedge, 0, 1\}$, $\{\vee, 0, 1\}$, or $\{\oplus, 0, 1\}$, respectively. This follows since these clones only contain constants, projections, and multi-ary versions of *not*, *and*, *or*, and $\oplus$, respectively.

Theorem 4.2. *Let N_- be a finite set of Boolean functions such that $[N_-] \subseteq \mathrm{N}$. Then $\mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$ is NL-complete.*

Proof. The lower bound follows from Lemma 4.1. For the upper bound, first note that for an LTL formula ψ the following equivalences hold: $\mathsf{FF}\psi \equiv \mathsf{F}\psi$, $\mathsf{GG}\psi \equiv \mathsf{G}\psi$, $\mathsf{FGF}\psi \equiv \mathsf{GF}\psi$, $\mathsf{GFG}\psi \equiv \mathsf{FG}\psi$, $\mathsf{G}\psi \equiv \neg\mathsf{F}\neg\psi$, and $\mathsf{F}\psi \equiv \neg\mathsf{G}\neg\psi$. Furthermore, it is possible to interchange X and adjacent G-, F-, or $\neg$-operators, yielding an equivalent formula. Under these considerations, each formula $\varphi \in \mathrm{L}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$ can be transformed without changing its satisfaction into a normal form

$$\varphi' = \mathsf{X}^m P \sim y\ ,$$

where P is a prefix ranging over the values "empty string", F, G, FG, and GF; m is the number of occurrences of X in φ; $\sim$ is either the empty string or $\neg$; and y is a variable or a constant. This normal

form has two important properties. First, it can be represented in logarithmic space using two binary counters a and b. The counter a stores m, and b takes on values $0, \ldots, 9$ to represent each possible combination of P and $\sim$. Note that a takes on values less than $|\varphi|$, and b has a constant range. Hence both counters require at most logarithmic space. It is not necessary to store any information about y, because it can be taken from the representation of φ.

Second, φ' can be *computed* from φ in logarithmic space. The value of a is obtained by counting the occurrences of X in φ, and b is obtained by linearly parsing φ with the automaton that is given in Figure 6, and which ignores all occurrences of X.

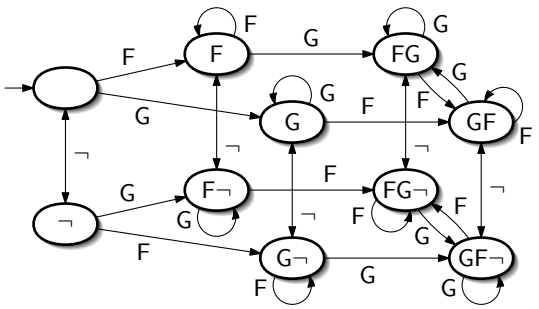

Fig. 6. An automaton that computes $P\sim$

The state of this automaton at the end of the passage through φ determines the values of P and $\sim$ in φ. Now let φ be an $\mathrm{L}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_{-})$-formula, $K = (W, R, \eta)$ a Kripke structure and $w \in W$. If y is constant, the problem is trivial, therefore it remains to consider the case where y is a variable. According to the possible values of P and $\sim$ in φ, there are ten cases to consider. We only present the argumentation for those five in which $\sim$ is empty. (For the dual cases, kindly replace each occurrence of "$\in \eta(b)$" by "$\notin \eta(b)$".) In the following list, we assume that $m = 0$. As per explanation below, this is not a significant restriction.

***P* is empty** Then $\langle \varphi, K, w \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_{-})$ if and only if $y \in \eta(w)$.

***P* = F** In this case we have to check whether there is a state $v \in W$ that can be reached from w via R, and $y \in \eta(v)$.

$\boldsymbol{P = \mathsf{G}}$ We define $W' = \{y \in W \mid y \in \eta(y)\}$ and $R' = R \cap W' \times W'$. It holds that $\langle \varphi, K, w\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$ if and only if there is some $v \in W'$ such that v is accessible from w via R' and v belongs to a cycle in R'.

$\boldsymbol{P = \mathsf{FG}}$ We can reduce this case to the previous one: $\langle \varphi, K, w\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$ if and only if there is some $v \in W'$ that can be reached from w via R, and $\langle \mathsf{G}y, K, v\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$.

$\boldsymbol{P = \mathsf{GF}}$ We have to check whether there exists some $v \in W$ that can be reached from w via R such that $y \in \eta(v)$ and v belongs to a cycle.

Since the questions whether there is a path from any vertex to another and whether any vertex belongs to a cycle in a directed graph can be answered in NL, all previously given procedures are NL-algorithms. The restriction $m = 0$ is removed by the observation that $\langle \mathsf{X}^m P{\sim}y, K, w\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$ if and only if there exists some state v in K that is accessible from w in m R-steps such that $\langle P{\sim}y, K, v\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$. This reduces the case $m > 0$ to $m = 0$.

Hence we have found an NL-algorithm deciding $\mathrm{MC}(\{\mathsf{F}, \mathsf{G}, \mathsf{X}\}, N_-)$: on input $\langle \varphi, K, w\rangle$, compute φ', guess a state v accessible from w in m R-steps, apply the procedure of one of the above five cases to $\langle \varphi', K, w\rangle$, and accept if the last step was successful. ❑

Theorem 4.3. *(1) Let V_- be a finite set of Boolean functions such that $[V_-] \subseteq \mathrm{V}$. Then $\mathrm{MC}(\{\mathsf{F}, \mathsf{X}\}, V_-)$ is NL-complete.*

(2) Let E_- be a finite set of Boolean functions such that $[E_-] \subseteq \mathrm{E}$. Then $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, E_-)$ is NL-complete.

Proof. The lower bounds follow from Lemma 4.1.

First consider the case $[V_-] \subseteq \mathrm{V}$. It holds that $\mathsf{F}(\psi_1 \vee \cdots \vee \psi_n) \equiv \mathsf{F}\psi_1 \vee \cdots \vee \mathsf{F}\psi_n$ as well as $\mathsf{XF}\varphi \equiv \mathsf{FX}\varphi$ and $\mathsf{X}(\varphi \vee \psi) \equiv \mathsf{X}\varphi \vee \mathsf{X}\psi$. Therefore, every formula $\varphi \in \mathrm{L}(\{\mathsf{F}, \mathsf{X}\}, V_-)$ can be rewritten as

$$\varphi' = \mathsf{FX}^{i_1} y_1 \vee \cdots \vee \mathsf{FX}^{i_n} y_n \vee \mathsf{X}^{i_{n+1}} y_{n+1} \vee \cdots \vee \mathsf{X}^{i_m} y_m,$$

where $y_1, \ldots, y_m$ are variables or constants (note that this representation of φ can be constructed in L). Now let $\langle \varphi, K, a\rangle$ be an instance of $\mathrm{MC}(\{\mathsf{F}, \mathsf{X}\}, V_-)$, where $K = (W, R, \eta)$, and let φ be of the

above form. Thus, $\langle \varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{X}\}, V_-)$ if and only if for some $j \in \{n+1, \ldots, m\}$, there is a state $b \in W$ such that $y_j \in \eta(b)$ and b is accessible from a in exactly i_j R-steps or if, for some $j \in \{1, \ldots, n\}$, there is a state $b \in W$ such that $y_j \in \eta(b)$ and b is accessible from a in at least i_j R-steps. This can be tested in NL.

As for the case $[E_-] \subseteq \mathrm{E}$, we take advantage of the duality of F and G, and $\wedge$ and $\vee$, respectively. Analogous considerations as above lead to the logspace computable normal form

$$\varphi' = \mathsf{G}\mathsf{X}^{i_1} y_1 \wedge \cdots \wedge \mathsf{G}\mathsf{X}^{i_n} y_n \wedge \mathsf{X}^{i_{n+1}} y_{n+1} \wedge \cdots \wedge \mathsf{X}^{i_m} y_m.$$

Let $I = \max\{i_1, \ldots, i_m\}$. For each $j = 1, \ldots, m$, we define $W^j = \{b \in W \mid y_j \in \eta(b)\}$ and $R^j = R \cap W^j \times W^j$. Furthermore, let W' be the union of W^j for $j = 1, \ldots, n$, and let $R' = R \cap W' \times W'$. Now $\langle \varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, E_-)$ if and only if there is some state $b \in W'$ satisfying the following conditions.

- There is an R-path p of length at least I from a to b, where the first $I + 1$ states on p are $c_0 = a$, c_1, …, c_I.
- The state b lies on a cycle in W'.
- For each $j = 1, \ldots, n$, each state of p from c_{i_j} to c_I is from W^j.
- For each $j = n + 1, \ldots, m$, the state c_{i_j} is from W^j.

These conditions can be tested in NL as follows. Successively guess $c_1, \ldots, c_I$ and verify their membership in the appropriate sets W^j. Then guess b, verify whether $b \in W'$, whether b lies on some R'-cycle, and whether there is an R'-path from c_I to b. ❑

In the proof of Theorem 4.3, we have exploited the duality of F and G, and $\vee$ and $\wedge$, respectively. Furthermore, the proof relied on the fact that F and $\vee$ (and G and $\wedge$) are interchangeable. This is not the case for F and $\wedge$, or G and $\vee$, respectively. Hence it is not surprising that $\mathrm{MC}(\{\mathsf{F}\}, \{\wedge\})$ is NP-hard (Corollary 3.3). However, the NL-membership of $\mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, \{\vee\})$ *is* surprising. Before we formulate this result, we try to provide an intuition for the tractability of this problem. The main reason is that an inductive view on $\mathrm{L}(\{\mathsf{F}, \mathsf{G}\}, \{\vee\})$-formulae allows us to subsequently guess parts of a satisfying path without keeping the previously guessed parts in memory. This is possible because each $\mathrm{L}(\{\mathsf{F}, \mathsf{G}\}, \{\vee\})$-formula φ can be

rewritten as

$$\varphi = y_1 \vee \cdots \vee y_n \vee \mathsf{F}z_1 \vee \cdots \vee \mathsf{F}z_m \vee \mathsf{G}\psi_1 \vee \cdots \vee \mathsf{G}\psi_\ell \vee \mathsf{FG}\psi_{\ell+1} \vee \cdots \vee \mathsf{FG}\psi_k, \tag{1}$$

where the y_i, z_i are variables (or constants), and each ψ_i is an $\mathrm{L}(\{\mathsf{F},\mathsf{G}\},\{\vee\})$-formula of the same form with a strictly smaller nesting depth of G-operators. Now, φ is *true* at the begin of some path p iff one of its disjuncts is *true* there. In case none of the y_i or $\mathsf{F}z_i$ is *true*, we must guess one of the $\mathsf{G}\psi_i$ (or $\mathsf{FG}\psi_j$) and check whether ψ_i (or ψ_j) is *true* on the entire path p (or on p minus some finite number of initial states). Now ψ_i is again of the above form. So we must either find an infinite path on which $y_1 \vee \cdots \vee y_n \vee \mathsf{F}z_1 \vee \cdots \vee \mathsf{F}z_m$ is *true* everywhere (a cycle containing at least $|N|$ states satisfying some y_i or z_i suffices, where N is the set of states of the Kripke structure), or we must find a *finite* path satisfying the same conditions and followed by an infinite path satisfying one of the $\mathsf{G}\psi_i$ (or $\mathsf{FG}\psi_j$) at its initial point. Hence we can recursively solve a problem of the same kind with reduced problem size. Note that it is neither necessary to explicitly compute the normal form for φ or one of the ψ_i, nor need previously visited states be stored in memory.

Theorem 4.4. *Let V_- be a finite set of Boolean functions such that $[V_-] \subseteq \mathrm{V}$. Then $\mathrm{MC}(\{\mathsf{G}\}, V_-)$ and $\mathrm{MC}(\{\mathsf{F},\mathsf{G}\}, V_-)$ are NL-complete.*

Proof. The lower bound follows from Lemma 4.1. It remains to show NL-membership of $\mathrm{MC}(\{\mathsf{F},\mathsf{G}\}, V_-)$. For this purpose, we devise the recursive algorithm $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$ as given in Table 2. Note that we have deliberately left out constants. This is no restriction, since we have observed in Lemma 2.2 that each constant can be regarded as a variable that is set to *true* or *false* throughout the whole Kripke structure.

The parameter *mode* indicates the current "mode" of the computation. The idea is as follows. In order to determine whether φ is satisfiable at the *initial* point of some structure starting at a in K, the algorithm has to be in mode `now`. This, hence, is the default setting for the first call of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$. As soon as the algorithm chooses to process a G-subformula $\mathsf{G}\alpha$ of φ, it has to determine whether α is satisfiable at *every* point in some structure starting at the currently

Algorithm $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathtt{V}}$

Input $\varphi \in \mathrm{L}(\{\mathsf{F},\mathsf{G}\}, V_{-})$
Kripke structure $K = (W, R, \eta)$
$a \in W$
additional parameter $mode \in \{\mathtt{now}, \mathtt{always}\}$

Output **accept** or **reject**

```
 1: c ← 0;   ψ ← φ;   b ← a;   Ffound ← false
 2: while c ≤ |W| do
 3:       if ψ = α0 ∨ α1  (for some α0, α1)  then
 4:             guess i ∈ {0, 1}
 5:             ψ ← αi
 6:       else if ψ = Fα  (for some α)  then
 7:             Ffound ← true
 8:             ψ ← α
 9:       else                    /* ψ is some Gα or a variable */
10:             if Ffound then /* process encountered F */
11:                   guess n with 0 ≤ n ≤ |W|
12:                   for i = 1, 2, ..., n do        /* if n = 0,
                      ignore this loop */
13:                         b ← guess some R-successor of b
14:                   end for
15:             end if
16:             if ψ = Gα  (for some α)  then
17:                   call MC{F,G},V(α, K, b, always)
18:             else                          /* ψ is a variable */
19:                   if ψ ∉ η(b) then
20:                         reject
21:                   end if
22:                   if mode = always then
23:                         c ← c + 1
24:                         b ← guess some R-successor of b
25:                         Ffound ← false
26:                         ψ ← φ                    /* in next
                            while-round, */
                                  /* check φ in new state b */
27:                   else
28:                         accept
29:                   end if
30:             end if
31:       end if
32: end while
33: accept
```

Table 2. The algorithm $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$

visited state in K. It therefore changes into `always` mode and calls itself recursively with the first parameter set to α, see Line 17.

Hence, given an instance $\langle \varphi, K, a \rangle$ of the problem $\mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-})$, we have to invoke $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{now})$ in order to determine whether there is a satisfying path for φ in K starting at a. It is easy to see that this call always terminates: First, whenever the algorithm calls itself recursively, the first argument of the new call is a strict subformula of the original first argument. Therefore there can be at most $|\varphi|$ recursive calls. Second, within each call, each passage through the *while* loop (Lines 2–32) either decreases φ or increases c. Hence, there can be at most $|\varphi| \cdot (|W| + 1)$ passages through the *while* loop until the algorithm accepts or rejects.

$\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$ is an NL algorithm: The values of all parameters and programme variables are either subformulae of the original formula φ, states of the given Kripke structure K, counters of range $0, \ldots, |W|+1$, or Booleans. They can all be represented using $\lceil \log |\varphi| \rceil$, $\lceil \log(|W|+1) \rceil$, or constantly many bits. Furthermore, the algorithm's recursive nature does not cause a problem: the recursive depth is linear, and no *return* command is used. Therefore no stack is needed at all. While it is possible to reformulate the algorithm avoiding recursion, this would make the following inductive correctness proof more difficult.

For the correctness of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$, we will proceed in two steps. We start by showing correctness in `always` mode, using induction on the nesting depth of G-operators in φ. We denote this value by $\mu_{\mathsf{G}}(\varphi)$. Claim 5 will then ensure the correct behaviour in `now` mode.

Claim 4. For each $\varphi \in \mathrm{L}(\{\mathsf{F}, \mathsf{G}\}, \mathrm{V})$, each $K = (W, R, \eta)$, and each $a \in W$:

$\langle \mathsf{G}\varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-}) \quad \Leftrightarrow \quad$ there is an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{always})$.

Proof of Claim 4. *For the base case of the induction,* let $\mu_{\mathsf{G}}(\varphi) = 0$. Because of the equivalences $\mathsf{F}(\psi_1 \vee \psi_2) \equiv \mathsf{F}\psi_1 \vee \mathsf{F}\psi_2$ and $\mathsf{FF}\psi \equiv \mathsf{F}\psi$, we may assume w.l.o.g. that any occurrence of the F-operator is in front of some variable in φ. If we think of φ as a tree, this means that F-operators can only occur in direct predecessors of leaves. Note that the algorithm computes this normal form implicitly: Whenever it guesses a path from the root (φ) to some leaf (a variable) in the tree and encounters an F-operator in Line 6, the flag *Ffound* is set.

Only after processing all $\vee$-operators on the remaining part of the path, the F-operator is processed in Lines 10–15. Now let $\mathrm{VAR}_1(\varphi)$ be all variables that occur in the scope of an F-operator in φ, and let $\mathrm{VAR}_0(\varphi)$ be all other variables in φ.

For the "$\Rightarrow$" direction, suppose $\langle \mathsf{G}\varphi, K, a\rangle \in \mathrm{MC}(\{\mathsf{F},\mathsf{G}\}, V_{\_})$. Then there exists a path π in K such that $\pi_0 = a$, and for all $i \geq 0$, $\pi, i \vDash \varphi$. This means that, for each i, either there exists some $x_i \in \mathrm{VAR}_0(\varphi)$ such that $\pi, i \vDash x_i$, or there is some $x_i \in \mathrm{VAR}_1(\varphi)$ such that $\pi, i \vDash \mathsf{F}x_i$. Now it can be seen that there is a non-rejecting sequence of runs through the *while* loop in Lines 2–32 after which c has value $|W|+1$, which then leads to the *accept* in Line 33:

Consider the begin of an arbitrary single run through the *while* loop in Line 2. Let p_i be the current value of b. If $x_i \in \mathrm{VAR}_0(\varphi)$, then the algorithm can "guess its way through the tree of φ" in Lines 3–5 and finally reaches Line 19 with $\psi = x_i$. It does not reject in Line 20, increases c in Line 23, guesses p_{i+1} in Line 24, and resets *Ffound* and ψ appropriately in Lines 25, 26. Otherwise, if $x_i \in \mathrm{VAR}_1(\varphi)$, then there is some $n \geq 0$ such that p_{i+n} satisfies x_i. It is safe to assume that $n \leq |W|$ because otherwise the path from p_i to p_{i+n} would describe a cycle within K which could be replaced by a shorter, more direct, path without affecting satisfiability of the relevant subformulae in the states $p_0, \ldots, p_i$. Now the algorithm can proceed as in the previous case, but, in addition, it has to guess the correct value of n and the sequence $p_{i+1}, \ldots, p_{i+n}$ in Lines 10–15.

For the "$\Leftarrow$" direction, let there be an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{always})$. Since the algorithm is in `always` mode, and φ is G-free, the acceptance can only take place in Line 33, without a recursive call in Line 17. Hence the counter c reaches value $|W|+1$ in the *while* loop in Lines 2–32.

Let $p = p_0, p_1, \ldots, p_m$ be the sequence of states guessed in this run in Lines 13 and 24, where $p_0 = a$. Furthermore, let $i_0, \ldots, i_{|W|+1}$ be an index sequence that determines a subsequence of p such that

- $0 = i_0 < i_1 < \cdots < i_{|W|+1} = m$, and
- for each $j > 0$, p_{i_j} is the value assigned to b in Line 24 after having set c to value j in Line 23.

Now it is clear that for all $j = 0, \ldots, |W|$, there must be a variable x_j such that $x_j \in \eta(p_{i_{j+1}-1})$. If $x_j \in \mathrm{VAR}_0(\varphi)$, then $p_{i_{j+1}} = p_{i_j} + 1$, and

each structure p' extending p beyond p_m satisfies x_j (and hence φ) at p_{i_j}. Otherwise $x_j \in \mathrm{VAR}_1(\varphi)$, and the accepting run of the algorithm has guessed the states $p_{i_j}, \ldots, p_{i_{j+1}-1}$ in Line 13. In this case, each structure p' extending p beyond p_m satisfies $\mathsf{F}x_j$ (and hence φ) at $p_{i_j}, \ldots, p_{i_{j+1}-1}$. From these two cases, we conclude that each such p' satisfies φ in all states $p_0, \ldots, p_m$.

We now restrict attention to the states $p_{i_1-1}, \ldots, p_{i_{|W|+1}-1}$. Among these $|W|+1$ states, some of the $|W|$ states of K has to occur twice. Assume p_{i_j-1} and p_{i_k-1} represent the same state from K, where $j < k$. Then we can create an (infinite) structure π from p that consists of states $p_0, \ldots, p_{i_k-1}$, followed by an infinite repetition of the sequence $p_{i_j}, \ldots, p_{i_k-1}$. It is now obvious that π satisfies φ in every state, hence $\pi, 0 \vDash \varphi$, that is, $\langle \mathsf{G}\varphi, K, a\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_-)$.

For the induction step, let $\mu_{\mathsf{G}}(\varphi) > 0$. For the same reasons as above, we can assume that any F-operator only occurs in front of variables or in front of some G-operator in φ. This "normal form" is taken care of by setting *Ffound* to `true` when F is found (Line 7) and processing this occurrence of F only when a variable or some G-operator is found (Lines 10–15).

For the "$\Rightarrow$" direction, suppose $\langle \mathsf{G}\varphi, K, a\rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_-)$. Then there exists a path π in K such that $\pi_0 = a$, and for all $i \geq 0$, $\pi, i \vDash \varphi$. We describe an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{always})$. Consider a single passage through the *while* loop with the following configuration. The programme counter has value 2, c has value at most $|W|$, b has value π_i, and ψ has value φ. Since $\pi \vDash \varphi$, there are four possible cases. The argumentation for the first two of them is the same as in the base case.

Case 1. $\pi, i \vDash x$, for some $x \in \mathrm{VAR}_0(\varphi)$.

Case 2. $\pi, i \vDash \mathsf{F}x$, for some $x \in \mathrm{VAR}_1(\varphi)$.

Case 3. $\pi, i \vDash \mathsf{G}\alpha$, for some maximal G-subformula $\mathsf{G}\alpha$ of φ that is *not* in the scope of some F-operator.
This means that α is *true* everywhere on the path $\pi_i, \pi_{i+1}, \pi_{i+2}, \ldots$. Hence, due to the induction hypothesis, $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\alpha, K, b_i, \mathtt{always})$ has an accepting run. By appropriate guesses in Line 4, the current call of the algorithm can reach that accepting recursive call in Line 17.

Case 4. $\pi, i \vDash \mathsf{G}\alpha$, for some maximal G-subformula $\mathsf{G}\alpha$ of φ that *is* in the scope of some F-operator.
By combining the arguments of Cases 3 and 2, we can find an accepting run for this case.

If only Cases 1 or 2 occur more than $|W|$ times in a sequence, then c will finally take on value $|W| + 1$, and this call will accept in Line 31. Otherwise, whenever one of Cases 3 and 4 occurs, than the acceptance of the new call—and hence of the current call—is due to the induction hypothesis.

For the "$\Leftarrow$" direction, let there be an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{always})$. Since the algorithm is in `always` mode, the acceptance can only take place in Line 33 or in the recursive call in Line 17. If the run accepts in Line 33, the same arguments as in the base case apply. If the acceptance is via the recursive call, then let $\pi = \pi_0, \ldots, \pi_m$ be the sequence of states guessed such that $\pi_0 = a$, and π_m is the value of b when the recursive call with $\mathsf{G}\alpha$ takes place. Due to the induction hypothesis, $\langle \mathsf{G}\alpha, K, b_m \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-})$ and, hence, there is an infinite structure π' extending π beyond π_m such that $\pi', m \vDash \mathsf{G}\varphi$. Furthermore, we can use the same argumentation as in the base case to show that, for each $i \leq m$, $\pi', i \vDash \varphi$. Therefore, $\pi', 0 \vDash \mathsf{G}\varphi$, which proves $\langle \mathsf{G}\varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-})$. ∎

Claim 5. For each $\varphi \in \mathrm{L}(\{\mathsf{F}, \mathsf{G}\}, V_{-})$, each $K = (W, R, \eta)$, and each $a \in W$:

$\langle \varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-}) \;\Leftrightarrow\;$ there is an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{now})$

Proof of Claim 5. *For the "$\Rightarrow$" direction,* suppose $\langle \varphi, K, a \rangle \in \mathrm{MC}(\{\mathsf{F}, \mathsf{G}\}, V_{-})$. Then there exists a path π in K such that $\pi_0 = a$ and $\pi, 0 \vDash \varphi$. We describe an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{now})$. Consider the first passage through the *while* loop with the following configuration. The programme counter has value 2, c has value 0 (this value does not change in *now* mode), b has value a, and ψ has value φ. Since $\pi, 0 \vDash \varphi$, there are four possible cases. The argumentation for them is very similar to that in the proof of Claim 4.

Case 1. $\pi, 0 \vDash x$, for some $x \in \mathrm{VAR}_0(\varphi)$.

As in the proof of Claim 4, the algorithm can guess the appropriate disjuncts in Lines 3–5, does not reject in Line 20 and accepts (it is in *now* mode!) in Line 28.

Case 2. $\pi, 0 \vDash \mathsf{F}x$, for some $x \in \mathrm{VAR}_1(\varphi)$.
As in the proof of Claim 4, there exists some n with $0 \leq n \leq |W|$ such that b_n satisfies x_i. The algorithm can proceed as in the previous case, but, in addition, it has to guess the correct value of n and the sequence $\pi_1, \ldots, \pi_n$ in Lines 10–15.

Case 3. $\pi, 0 \vDash \mathsf{G}\alpha$, for some maximal G-subformula $\mathsf{G}\alpha$ of φ that is *not* in the scope of some F-operator.
This means that α is *true* everywhere on the path π. Hence, due to the induction hypothesis, $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\alpha, K, b_i, \mathtt{now})$ has an accepting run. By appropriate guesses in Line 4, the current call of the algorithm can reach that accepting recursive call in Line 17.

Case 4. $\pi, 0 \vDash \mathsf{G}\alpha$, for some maximal G-subformula $\mathsf{G}\alpha$ of φ that *is* in the scope of some F-operator.
By combining the arguments of Cases 3 and 2, we can find an accepting run for this case.

For the "$\Leftarrow$" direction, suppose there is an accepting run of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}(\varphi, K, a, \mathtt{now})$. Since the algorithm is in `now` mode, the acceptance can only take place in Line 28 or in the recursive call in Line 17. If the run accepts in Line 28, then there is some variable x such that either $x \in \mathrm{VAR}_0(\varphi)$ and $x \in \eta(a)$, or $x \in \mathrm{VAR}_1(\varphi)$ and the run guesses a path $\pi_0, \ldots, \pi_m$ with $\pi_0 = a$ and $x \in \eta(\pi_m)$. In both cases, each structure π' extending the sequence of states guessed so far, satisfies φ at a. On the other hand, if the run accepts in the recursive call, we can argue as in the proof of Claim 4. ∎ ❑

Unfortunately, the above argumentation fails for $\mathrm{MC}(\{\mathsf{G}, \mathsf{X}\}, V)$ because of the following considerations. The NL-algorithm in the previous proof relies on the fact that a satisfying path for $\mathsf{G}\psi$, where ψ is of the form (1), can be divided into a "short" initial part satisfying the disjunction of the atoms, and the remaining end path satisfying one of the $\mathsf{G}\psi_i$ at its initial state. When guessing the initial part, it suffices to separately guess each state and consult η.

Algorithm $\mathtt{MC}_{\{\mathtt{X}\},\mathtt{L}}$

Input
$\varphi' = \mathsf{X}^{i_1} p_1 \oplus \cdots \oplus \mathsf{X}^{i_\ell} p_\ell$
Kripke structure $K = (W, R, \eta)$
$a \in W$

Output
accept or **reject**

```
1: parity ← 0;  b ← a;  k ← 0
2: while k ≤ m do
3:       for j = 1, ..., ℓ do
4:             if i_j = k and p_j ∈ η(b)
               then
5:                   parity ← 1 − parity
6:             end if
7:       end for
8:       k ← k + 1
9:       b ← guess some R-successor of b
10: end while
11: if parity = 1 then
12:       accept
13: else
14:       reject
15: end if
```

Table 3. The algorithm $\mathtt{MC}_{\{\mathtt{X}\},\mathrm{L}}$

If X were in our language, the disjuncts would be of the form $\mathsf{X}^{k_i} y_i$ and $\mathsf{X}^{\ell_i}\mathsf{G}\psi_i$. Not only would this make the guessing of the initial part more intricate. It would also require memory for processing each of the previously satisfied disjuncts $\mathsf{X}^{k_i} y_i$. An adequate modification of $\mathtt{MC}_{\{\mathsf{F},\mathsf{G}\},\mathrm{V}}$ would require more than logarithmic space. We have shown NP-hardness for $\mathrm{MC}(\{\mathsf{G},\mathsf{X}\}, V)$ in Theorem 3.4.

Theorem 4.5. *Let L_- be a finite set of Boolean functions such that $[L_-] \subseteq \mathrm{L}$. Then $\mathrm{MC}(\{\mathsf{X}\}, L_-)$ is NL-complete.*

Proof. The lower bound follows from Lemma 4.1.

For the upper bound, let $\varphi \in \mathrm{L}(\{\mathsf{X}\}, L_-)$ be a formula, $K = (W, R, \eta)$ a Kripke structure, and $a \in W$ a state. Let m denote the maximal nesting depth of X-operators in φ. Since for any k-ary Boolean operator f from L_-, the formula $\mathsf{X} f(\psi_1, \ldots, \psi_k)$ is equivalent to $f(\mathsf{X}\psi_1, \ldots \mathsf{X}\psi_k)$, φ is equivalent to a formula $\varphi' \in \mathrm{L}(\{\mathsf{X}\}, L_-)$ of the form

$$\varphi' = \mathsf{X}^{i_1} p_1 \oplus \cdots \oplus \mathsf{X}^{i_\ell} p_\ell \,,$$

where $0 \leq i_j \leq m$ for each $j = 1, \ldots, \ell$. It is not necessary to compute φ' all at once, because it will be sufficient to calculate i_j each time the variable p_j is encountered in the algorithm $\mathtt{MC}_{\{\mathsf{X}\},\mathrm{L}}$ given in Table 3.

It is easy to see that $\mathtt{MC}_{\{\mathsf{X}\},\mathrm{L}}$ accepts if and only if φ is satisfiable in state a of K. From the used variables, it is clear that $\mathtt{MC}_{\{\mathsf{X}\},\mathrm{L}}$ runs in nondeterministic logarithmic space. ❑

5 Conclusion, and open problems: the ugly fragments

We have almost completely separated the model-checking problem for Linear Temporal Logic with respect to arbitrary combinations of propositional and five future-looking temporal operators into tractable and intractable cases. We have shown that all tractable MC problems are NL-complete or even easier to solve. This exhibits a surprisingly large gap in complexity between tractable and intractable cases. The only fragments that we have not been able to cover by our classification are those where only the binary *xor*-operator is allowed. However, it is not for the first time that this constellation has been difficult to handle, see [**?**, **?**]. Therefore, these fragments can justifiably be called ugly.

The borderline between tractable and intractable fragments is somewhat diffuse among all sets of temporal operators without U. On the one hand, this borderline is not determined by a single set of propositional operators (which is the case for the satisfiability problem, see [**?**]). On the other hand, the columns E and V do not, as one might expect, behave dually. For instance, while $\mathrm{MC}(\{\mathsf{G}\},\mathrm{V})$ is tractable, $\mathrm{MC}(\{\mathsf{F}\},\mathrm{E})$ is not—although F and G are dual, and so are V and E.

While it would be nice to complement our picture with past operators and find out whether the general statement "past is for free" made in [**?**] still holds for sub-Boolean fragments of LTL, a complete classification with future *and* past operators will significantly blow up the number of fragments to consider: instead of $2^5-1=31$ sets of temporal operators, we would have to consider $2^{10}-1=1023$ sets, each combined with 7 sets of Boolean operators. This extensive analysis would be of quadratic size compared to the current one. Given the large number of NL-complete fragments without past operators, whose upper bounds do not straightforwardly extend to their extensions by past operators, a significant additional number of theorems

would have to be proven. So far, we can only say that almost all fragments containing the S (since) operator are as hard as the corresponding fragments with U, for the same reasons. However, there are exceptions which are due to the asymmetry that paths have a first, but no last, state. These results can be found in slightly more detail in the conference version [**?**] of this article. Also, to keep the paper manageable, we only considered existential model checking, as in many previously published papers [**?**, **?**, **?**]. We defer a systematic treatment of universal model checking to a future study.

Further work should find a way to handle the open *xor* cases from this paper as well as from [**?**, **?**]. In addition, the precise complexity of all hard fragments not in bold-face type in Table 1 could be determined. Furthermore, we find it a promising perspective to use our approach for obtaining a fine-grained analysis of the model-checking problem for more expressive logics, such as CTL, CTL*, and hybrid temporal logics. Some results in this direction can be found in [**?**, **?**].

6 Acknowledgement

We thank the anonymous referees for their valuable comments.