The Computational Complexity Column

by

V. Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

Mathematical logic and computational complexity have close connections that can be traced to the roots of computability theory and the classical decision problem. In the context of complexity, some well-known fundamental problems are: satisfiability testing of formulas (in some logic), proof complexity, and the complexity of checking if a given model satisfies a given formula. The Model Checking problem, which is the topic of the present article, is also of practical relevance since efficient model checking algorithms for temporal/modal logics are useful in formal verification.

In their excellent and detailed survey, Arne Meier, Martin Mundhenk, Julian-Steffen Müller, and Heribert Vollmer tell us about the complexity of model checking for various logics: temporal, modal and hybrid and their many fragments. Their article brings out the intricate structures involved in the reductions and the effectiveness of standard complexity classes in capturing the complexity of model checking.

Complexity of Model Checking for Logics over Kripke models

Arne Meier* Julian-Steffen Müller*
Martin Mundhenk† Heribert Vollmer*

1 Introduction

Model checking refers to the task to verify automatically that a given software or hardware system fulfills certain specifications. The system is typically a finite state system, for example a hardware system or a concurrent program given in a restricted programming language (in general, the problem is of course undecidable), and the specification is given as a logical formula in some temporal logic [6]; typically properties that one wants to verify are safety properties such as livelihood or deadlock-freeness. The history of model checking is a remarkable success story, see, e.g., [7] or the other contributions in [14], and many tools exist and are used in practice.

In theoretical studies, the system is a Kripke model, i.e., a labelled transition graph, and the logical languages are variants of modal logics (basic modal logic, different temporal logics, modal dependence logic, etc.). That is, the following problem is studied:

Problem:	$\mathcal{L}$-MC
Description:	the model checking problem for logical language $\mathcal{L}$.
Input:	An $\mathcal{L}$-formula ϕ, a Kripke model $\mathcal{M}$, and an initial state w_0 in $\mathcal{M}$.
Question:	Does ϕ hold in w_0?

This paper addresses the computational complexity of model checking. This issue is covered in a very substantial survey paper by Philippe Schnoebelen [32], where upper bounds in form of algorithms and lower bounds in

*Leibniz Universität Hannover, Institut für Theoretische Informatik, {meier, mueller, vollmer}@thi.uni-hannover.de.

†Friedrich-Schiller-Universität Jena, Institut für Informatik, Martin.Mundhenk@uni-jena.de.

terms of hardness results for some basic complexity classes for the maybe most important temporal logics LTL, CTL, and CTL*.

It is well-known that the computational hardness of logical problems does not always depend on formulas of the full language but appears already for fragments. For example, the propositional satisfiability problem was shown to be NP-hard for formulas that use the negation of implication as only operator, i.e., as only logical connective [19]. More abstractly, Lewis [19] showed that the satisfiability problem for formulas with arbitrary Boolean operators is NP-complete if and only if the used operators allow to express the negation of implication. Corollaries of this are the well-known facts that satisfiability for monotone or for affine formulas is efficiently solvable.

As another example let us consider the maybe most basic model checking problem, namely for propositional logic. A model here is simply an assignment, and model checking is the problem to check if a given assignment satisfies a given formula—hence propositional model checking is the formula value problem. It is well-known that this problem is NC^1-complete for unrestricted formulas [4]. Henning Schnoor proved that it remains complete when restricted to monotone formulas, more generally he showed that propositional model checking is NC^1-complete if and only if the used operators allow to express $\wedge$ and $\vee$ [33].

In the present paper we will survey the computational complexity of model checking for modal logic and extensions like hybrid logic, intuitionistic logic, temporal logics (LTL, CTL, CTL*), and modal dependence logic. We will put particular emphasis on the study of the complexity of fragments of the logical language. After a preliminary section, where we lay the ground for a systematic study of fragments and introduce a basic propositional language, we treat the different modal logics in the above order in the subsequent sections. The results on basic modal logic are new (but easily obtained from the corresponding results for CTL, treating $\Box$ like AX and $\Diamond$ like EX). The classification for CTL fragments with exactly one temporal operator is new. The result on CTL* is new. All other results can be found already in the literature and we provide complete references.

2 Preliminaries

The structure of the Boolean clones. As explained before, the present paper will survey complexity classifications for fragments of modal logic and extensions, where the fragments are defined by restricting the set of allowed propositional connectives and modal operators. Post [28] classified the lattice of all relevant sets of Boolean operators—called *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 [3]. Whereas in general there is an infinite set of clones, for model checking luckily there are only seven different clones [1]. This is essentially due to the fact that the constants for *false* and *true* do not need to be part of the language but can be expressed by atoms that are either nowhere or everywhere satisfied in the model, and there are only seven clones that contain the constants. This means (and is proved formally in [37]) that if one wishes to study the computational complexity of model checking for propositional formulas with logical connectives restricted to some set B of Boolean functions, it is not necessary to consider all infinite possibilities for such sets B but actually suffices to consider these seven clones, depicted in Figure 1, where we describe the clones by their standard bases (we use $\oplus$ to denote the exclusive or). Notice that even though $\{\wedge, \oplus\}$ is not a base for all Boolean functions, it suffices to express all Boolean functions w.r.t. model checking problems because of the "free" existence of the constant *false*; e.g., $\neg x = x \oplus 1$ and $x \vee y = ((x \oplus 1) \wedge (y \oplus 1)) \oplus 1$.

To summarize, given any finite set B of Boolean functions/propositional connectives, the computational complexity of model checking for formulas (from any of the modal logics in the sequel) using propositional connectives from B is equivalent to the complexity of model checking for one of the bases given in Figure 1. To give one further example, a reader might wonder about formulas involving propositional implication $\rightarrow$. From [28] it follows that with $\rightarrow$ plus constant false, we can already express all Boolean functions (this can easily be verified since $\neg x \equiv x \rightarrow 0$ and $x \vee y \equiv (x \rightarrow 0) \rightarrow y$), hence model checking for formulas over $\{\rightarrow\}$ is equivalent to model checking for formulas over $\{\wedge, \oplus\}$.

Hence, in all upcoming results, if we classify the computational complexity of a model checking problem for the bases in Figure 1, we have in fact achieved a full complexity classification for all finite sets B of Boolean connectives.

The basic language. A language for a logic is a set of formulas of the form

$$\begin{aligned} \varphi \quad ::= \quad & \circ_{f_1}(\varphi, \ldots, \varphi) \mid \cdots \mid \circ_{f_k}(\varphi, \ldots, \varphi) \mid \\ & g_1(a_1, \ldots, a_{n_1}) \mid \cdots \mid g_\ell(a_1, \ldots, a_{n_\ell}), \end{aligned}$$

where $\circ_{f_i}$ are operators that are applied on formulas (like $\neg$, $\wedge$), and g_i are simple operators applied on atoms $a_j \in \mathrm{PROP}$ only (like constants, atoms, atomic negation, dependence atoms (see Section 8)).

Our basic language is the language of propositional logic with the identity *id* as simple operator (this is a little strange way to say that atoms are

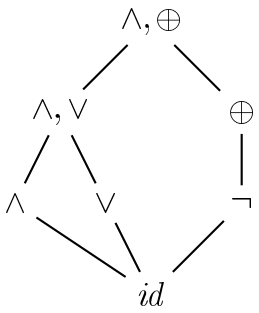


Figure 1: The Boolean clones relevant for model checking, represented by their standard bases. *id* denotes the clone represented without operator ("identity" of an atom).

formulas) and with the operators used in the bases for the relevant clones, namely the unary operator $\neg$ and the binary operators $\wedge, \vee, \oplus$. We use infix notation for binary operators and we do not write out *id*, as usual.

Kripke models. We will consider different propositional logics whose formulas base on a countable set PROP of *propositional atoms*. A *Kripke model* is a triple $\mathcal{M} = (U, R, \xi)$, where U is a nonempty and finite set of *states*, R is a binary relation on U — the *successor relation* — and $\xi : \text{PROP} \to \mathfrak{P}(U)$ is a function — the *valuation function*. For any atom it assigns the set of states in which this atom is satisfied. (U, R) can also be seen as a directed graph—it is called a *frame* in this context. A frame (U, R) is *reflexive*, if $(x, x) \in R$ for all $x \in U$, and it is *transitive*, if for all $a, b, c \in W$, it follows from $(a, b) \in R$ and $(b, c) \in R$ that $(a, c) \in R$. It is *total*, if for all $a \in U$ exists some $b \in U$ with $(a, b) \in R$, i.e. every state has a successor.

We present Kripke models graphically as directed graphs, where atom a is written into the node that represents state s iff $s \in \xi(a)$. As example, Figure 2 shows a Kripke model K_C that has two states to which the atom t is assigned, and one state that is named *out*.

The semantics of the basic language and its considered extensions is defined via Kripke models. Given a Kripke model $\mathcal{M} = (U, R, \xi)$ and a state $s \in U$, the *satisfaction relation* $\models$ is defined as follows.

$$
\begin{array}{lll}
\mathcal{M}, s \models p & \text{iff} & s \in \xi(p),\ p \in \text{PROP}, \\
\mathcal{M}, s \models \neg\varphi & \text{iff} & \mathcal{M}, s \not\models \varphi, \\
\mathcal{M}, s \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, s \models \varphi \text{ and } \mathcal{M}, s \models \psi, \\
\mathcal{M}, s \models \varphi \vee \psi & \text{iff} & \mathcal{M}, s \models \varphi \text{ or } \mathcal{M}, s \models \psi, \\
\mathcal{M}, s \models \varphi \oplus \psi & \text{iff} & \mathcal{M}, s \models \varphi \text{ and } \mathcal{M}, s \not\models \psi, \text{ or} \\
 & & \quad \mathcal{M}, s \not\models \varphi \text{ and } \mathcal{M}, s \models \psi.
\end{array}
$$

This definition is made so overly general because we want to define the languages of modal, temporal, and dependence logics as extensions of the basic language by more operators and its semantics by adding rules to the satisfaction relation for the additional operators. For this, we also need a semantics that is defined on paths through a Kripke model instead on states. An infinite path $\pi = x_1, x_2, \ldots$ through a Kripke model $\mathcal{M} = (U, R, \xi)$ is an infinite sequence of states $x_i \in U$ such that $(x_i, x_{i+1}) \in R$ holds for all $i \geq 1$. Whenever we write $\pi[i]$ we refer to the ith state in π, i.e., $\pi[i] := x_i$. Further π^i denotes the suffix of π starting at position i, i.e., $\pi^i := \pi[i], \pi[i+1], \pi[i+2], \ldots$. If $w \in U$ then $\Pi(w)$ is the set of all paths starting in w, i.e., for all $\pi \in \Pi(w)$ it holds that $\pi[1] = w$. For a formula φ of the basic language, a Kripke model $\mathcal{M}$ and a path π through $\mathcal{M}$, the satisfaction relation is defined as follows.

$$\mathcal{M}, \pi \models \varphi \quad \text{iff} \quad \mathcal{M}, \pi[1] \models \varphi.$$

Complexity. We use the following complexity classes and complete problems.

NC^1 is the class of problems decided logspace-uniform families of circuits consisting of $\wedge$-gates and $\vee$-gates with fan-in 2 and $\neg$-gates, that have logarithmic depth. A typical complete problem is the formula evaluation problem for propositional logic.

NL is the class of problems decided by nondeterministic logarithmically space bounded Turing machines. The typical complete problem is the graph accessibility problem for directed graphs REACH (given a directed graph with two nodes s and t, is there a path form s to t?).

LOGCFL is the class of problems decided by nondeterministic logarithmically space bounded Turing machines, that are additionally allowed to use a stack and run in polynomial time. The typical complete problem is the evaluation problem for monotone SAC^1 circuits. That are circuits of $\wedge$-gates, $\vee$-gates, 1-gates and 0-gates, where the $\wedge$-gates have indegree 2, the $\vee$-gates have arbitrary indegree, and the depth of the circuit is logarithmic in its size.

AC^1 is the class of problems decided by alternating logarithmically space bounded Turing machines with logarithmically bounded number of alternations. The typical complete problem is the evaluation problem for monotone AC^1 circuits, that are like SAC^1 circuits but with $\wedge$-gates having unbounded indegree.

P is the class of problems decided by polynomially time bounded Turing machines. A typical complete problem is the evaluation problem for monotone circuits.

NP is the class of problems decided by nondeterministic polynomially time bounded Turing machines. A typical complete problem is CNF-SAT that asks whether a given Boolean formula in conjunctive normal form is satisfiable. 3SAT is the restriction of CNF-SAT to formulas that have at most three literals in every clause.

$\mathrm{P}^{\mathrm{NP[1]}}$ is the class of problems decided by deterministic polynomially time bounded oracle Turing machines that make only one query to an NP-oracle.

Θ_2^{P} is the class of problems decided by deterministic polynomially time bounded oracle Turing machines with an NP-oracle, where the number of oracle queries is logarithmically bounded. A typical complete problem is ODDS that asks for a given sequence $\varphi_1, \varphi_2, \ldots, \varphi_n$ of Boolean formulas, whether there exists an odd i such that $\varphi_1, \ldots, \varphi_i$ are satisfiable and φ_{i+1} is not satisfiable [41].

PSPACE is the class of problems decided by polynomially space bounded Turing machines. A typical complete problem is the value problem for quantified Boolean formulas QBF-VAL. This problem has instances $\exists x_1 \forall x_2 \exists x_3 \ldots \forall x_{n-1} \exists x_n \varphi$, where φ is an instance of 3SAT with variables $x_1, \ldots, x_n$ for odd n. An instance belongs to QBF-VAL if there exists $z_1 \in \{\bot, \top\}$ such that for all $z_2 \in \{\bot, \top\} \ldots$ $\varphi[x_1/z_1] \ldots [x_n/z_n]$ evaluates to $\top$, where $\bot$ and $\top$ stand for *false* and *true*. The formula evaluation problem for first-order logic is another PSPACE-complete problem.

3 Modal logic and hybrid logic

Modal logic can be seen as the simplest extension of Boolean logic that allows easily to simulate the behaviour of computer programs in the means of different program states. It has been firstly introduced by Lewis [18] and has become very popular since Kripke [16] found its intuitive semantics. One of the standard textbooks on modal logic is [5].

The language of modal logic extends the basic language by the unary operators $\Diamond$ and $\Box$. The semantics of these operators is as follows. Let $\mathcal{M} = (W, R, \xi)$ be a Kripke model with $s \in W$.

$$\begin{array}{lll} \mathcal{M}, s \models \Diamond\varphi & \text{iff} & \exists t \in W \text{ with } (s,t) \in R : \ \mathcal{M}, t \models \varphi, \\ \mathcal{M}, s \models \Box\varphi & \text{iff} & \forall t \in W \text{ with } (s,t) \in R : \ \mathcal{M}, t \models \varphi. \end{array}$$

The model checking problem for modal logic is defined as follows.

Problem:	ML-MC(O)
Description:	the model checking problem for modal logic.
Input:	A modal formula ϕ using operators in $O \subseteq \{\Diamond, \Box, \wedge, \vee, \neg, \oplus\}$, a Kripke model $\mathcal{M} = (W, R, \xi)$, and an initial state $w_0 \in W$.
Question:	Does $\mathcal{M}, w_0 \models \phi$ hold?

Fischer and Ladner [12] showed that model checking for modal logic is in P. The algorithm essentially uses a dynamic programming approach to check inductively over all subformulas, in which states of the model they are satisfied. One can see it as if the states are marked inductively with the subformulas that they satisfy. Because the modal operators only take neighboured states into account, the algorithm runs in polynomial time.

Theorem 3.1 *[12]* ML-MC($\Diamond, \Box, \wedge, \vee, \neg, \oplus$) *is in* P.

Model checking for the fragments without any modal operator is the same as evaluating propositional formulas, that is at most NC^1-complete (see [33]). For the full modal logic, the complexity rises to P-completeness, what can be seen as a folklore result. This hardness is due to the "power" of the modal operator $\Box$ and its dual $\Diamond$, and independent of propositional operators.

Theorem 3.2 ML-MC($\Diamond, \Box$) *is* P*-complete.*

Proof. We give a reduction from the P-complete monotone circuit evaluation problem. Given a monotone circuit C and an input x. Assume the circuit to be layered alternatingly having an or-gate *out* as output gate. Let ℓ be the depth (i.e. the number of layers) of C. For simplicity, assume that ℓ is even. The Kripke model K_C is constructed by reversing all edges of the circuit C and letting t be satisfied in all input-nodes with input 1. An example of this construction can be seen in Figure 2. Then it holds that C outputs 1 if and only if $K_C, \mathit{out} \models (\Diamond\Box)^{\frac{\ell}{2}} t$. Notice that the use of the atom t can be omitted, if we have the constant $\bot$ (*false*) in our language. If all input-nodes with input 0 get an edge to itself, then the formula $(\Diamond\Box)^{\frac{\ell}{2}}\Box\bot$ simulates the evaluation of the circuit. □

The complexity of fragments with only one modal operator shows an interesting structure. If we have one modal operator and negation, then we can simulate the other modal operator and obtain P-hardness as in the above proof of Theorem 3.2. Say that $\Diamond$ has a *disjunctive* flavour, because $\Diamond\alpha$ is satisfied in a state w if w has *some* neighbour in which α is satisfied. Accordingly, $\Box$ has a *conjunctive* flavour. Combinations of a modal operator and its in this flavour "dual" operator on the Boolean side (i.e. $\Diamond$ and $\wedge$,

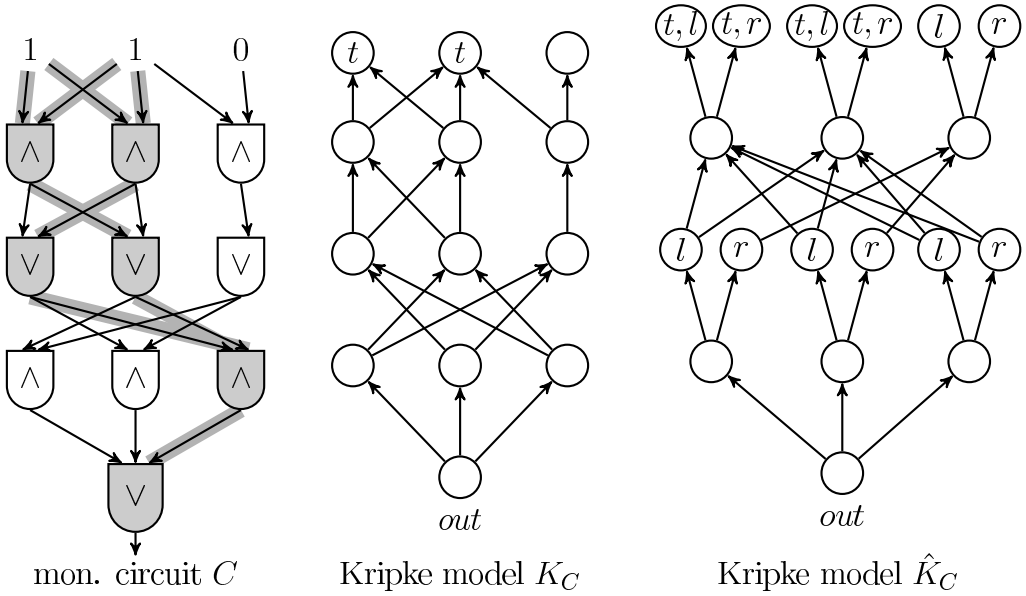


Figure 2: A monotone Boolean circuit C (the gates that output 1 are marked grey), and two Kripke model K_C (proof of Theorem 3.2) and $\hat{K}_C$ (proof of Theorem 3.3(2)) constructed from it.

or $\Box$ and $\vee$) yield a harder model checking problem than combinations of a modal operator with a Boolean operator of the same flavour.

Theorem 3.3 *Let* $B \subseteq \{\wedge, \vee, \neg, \oplus\}$. *Then* ML-MC$(\Diamond, B)$ *is*

1. P-*complete if* $B \supseteq \{\neg\}$ *or* $B \supseteq \{\oplus\}$,
2. LOGCFL-*complete if* $B = \{\wedge\}$ *or* $B = \{\wedge, \vee\}$,
3. NL-*complete otherwise.*

Proof. 1. Since $\Box\alpha \equiv \neg\Diamond\neg\alpha$, this follows from the proof of Theorem 3.2.

2. We reduce the evaluation problem for monotone SAC^1 circuits to the modal logic model checking problem for $\{\Diamond, \wedge\}$-formulas. This is essentially the same reduction as for a CTL model checking problem in [2, Prop.3.7]. Assume that the circuit is layered, has an or-gate as output-gate (layer 0), input gates at layer ℓ (that are also considered as or-gates), and and-gates at layer $\ell - 1$. The SAC^1 circuits has or-gates with arbitrarily many inputs and and-gates with 2 inputs each.

The nodes of the corresponding graph consist of the output-gate *out*, a node for every and-gate of C, $V_\wedge$, and a node for every edge into an and-gate of C. This essentially makes in G as many copies of every or-gate in C as the

or-gate's out-degree in C is. The edges of G are essentially the reversed edges of C. In order to be able to distinguish both the inputs to the and-gates, the corresponding nodes are assigned different atoms l (like *left*) and r. Nodes that correspond to 1-input gates are assigned to the atom t. An example of this construction can be seen as Kripke model $\hat{K}_C$ in Figure 2. Now define recursively $\{\Diamond, \wedge\}$-formulas $(\phi_i)_{0 \leq i \leq \ell}$ by

$$\phi_i := \begin{cases} t, & \text{if } i = \ell, \\ \Diamond \phi_{i+1}, & \text{if layer } i \text{ consists of or-gates,} \\ \Diamond(l \wedge \phi_{i+1}) \wedge \Diamond(r \wedge \phi_{i+1}), & \text{if layer } i \text{ consists of and-gates.} \end{cases}$$

Since an or-gate outputs 1, if at least one of its inputs equals 1, this can be simulated with a $\Diamond$. An and-gate outputs 1, if both of its inputs equal 1. This is simulated by checking both the left input and the right input in the Kripke model. Since this doubles the size of the formula, it is essential that the circuit has logarithmic depths only. Therefore the construction of the modal model checking instance can be done in logarithmic space. Then it is not hard to see that circuit C outputs 1 if and only if $\hat{K}_C, \mathit{out} \models \phi_0$.

For membership in LOGCFL save the subformulas and corresponding states on the stack. If the outer-most operator is a $\Diamond$, then guess a successor state in the Kripke model. If it is a $\vee$, then guess the subformula to continue with. If it is an $\wedge$, then push both subformulas plus the recent state on the stack. If it is a propositional formula, then evaluate it (in $\mathrm{NC}^1 \subseteq \mathrm{L}$) and reject if it is not true in this state. Eventually the stack becomes empty and we accept.

3. NL-hardness is shown for $\{\Diamond\}$-formulas using a usual reduction from the NL-complete directed graph accessibility problem. For membership in NL for $\{\Diamond, \vee\}$-formulas, notice that $\Diamond(\alpha \vee \beta) \equiv (\Diamond\alpha \vee \Diamond\beta)$. Thus simply guess which subformula is satisfied on which path. □

In the same way, we get a complete characterization of fragments with $\Box$.

Theorem 3.4 *Let* $B \subseteq \{\wedge, \vee, \neg, \oplus\}$. *Then* ML-MC$(\Box, B)$ *is*

1. P-*complete if* $B \supseteq \{\neg\}$ *or* $B \supseteq \{\oplus\}$,
2. LOGCFL-*complete if* $B = \{\wedge, \vee\}$ *or* $B = \{\vee\}$,
3. NL-*complete otherwise.*

Theorems 3.3, 3.4, and 3.2 now yield a complete characterization of the complexity of the model checking problems for all modal fragments in terms of completeness for NL, LOGCFL, resp. P—see Figure 3. This is a wonderful

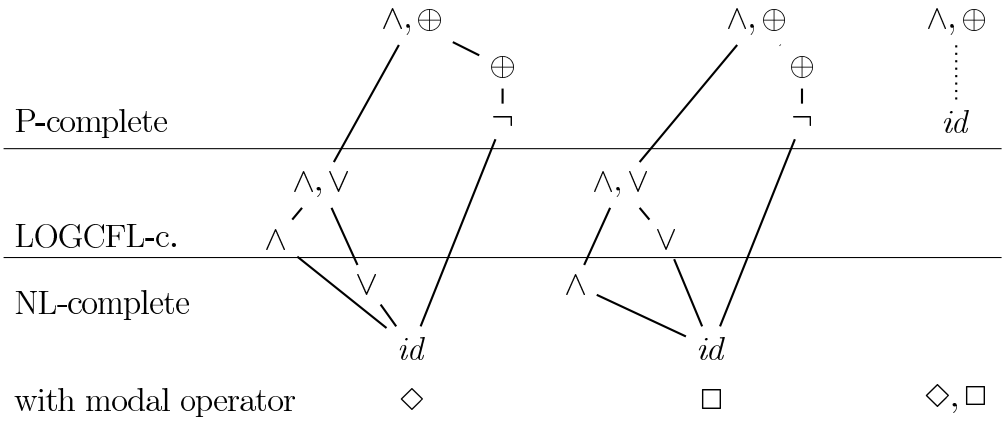


Figure 3: Overview over the completeness results for the complexity of model checking for fragments of modal logics from Theorems 3.3, 3.4, and 3.2.

situation that we unfortunately do not reach for the other logics that we consider here.

Hybrid logics are extensions of modal logic that allow for naming and accessing states of a model explicitly. As well as the foundations of temporal logic, it goes back to Prior [30]. We will only consider the hybrid operators $\downarrow$, that allows to give names to states in the Kripke model, and @, that allows to jump to a state with such a name. Because the notation of hybrid operators is a little different from the usual notation, we give a full definition of the language of hybrid logic, that we will use here.

$$\varphi \quad ::= \quad p \mid \mathsf{s} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \oplus \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \downarrow \mathsf{s}.\varphi \mid @_{\mathsf{s}}\varphi\,,$$

where p is a propositional atom and s is a state variable. The semantics uses a Kripke model $\mathcal{M} = (W, R, \xi)$ and additionally needs an assignment function g that maps each state variable to exactly one state—other than ξ, that maps every atom to an arbitrary set of states. Let g^x_{s} be the same as g, but with $g(\mathsf{s}) = x$.

$$\begin{array}{lll} \mathcal{M}, g, w \models \mathsf{s} & \text{iff} & g(\mathsf{s}) = w, \text{ if } \mathsf{s} \text{ is a state variable,} \\ \mathcal{M}, g, w \models \downarrow \mathsf{s}.\varphi & \text{iff} & \mathcal{M}, g^w_{\mathsf{s}}, w \models \varphi, \\ \mathcal{M}, g, w \models @_{\mathsf{s}}\varphi & \text{iff} & \mathcal{M}, g, g(\mathsf{s}) \models \varphi. \end{array}$$

The semantics of the modal and Boolean operators and of the propositional atoms is defined as for modal logic—the g makes no difference. The model checking problem for modal logic is defined as follows.

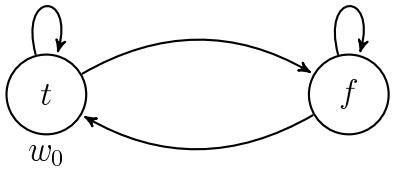


Figure 4: The Kripke model used for the PSPACE-completeness of HL-MC($\downarrow, \Diamond, \Box, \wedge, \vee$) in the proof of Theorem 3.6.

Problem:	HL-MC(O)
Description:	the model checking problem for hybrid logic.
Input:	A hybrid formula ϕ with operators in $O \subseteq \{\downarrow, @, \Box, \Diamond, \wedge, \vee, \neg, \oplus\}$, a Kripke model $\mathcal{M} = (W, R, \xi)$, an assignment g, and an initial state $w_0 \in W$.
Question:	Does $\mathcal{M}, g, w_0 \models \phi$ hold?

In general, model checking for hybrid logics even with some more hybrid operators can be done by a search algorithm that searches through all possibilities.

Theorem 3.5 *[13]* HL-MC($\downarrow, @, \Box, \Diamond, \wedge, \vee, \neg, \oplus$) *is in* PSPACE.

If we only add the @-operator to the modal language and leave out the $\downarrow$-operator, it is not hard to see that the model checking problem remains in P. What makes hybrid logic so powerful is the possibility to give names to states. The maximal hardness is reached already by adding the binder $\downarrow$ to the monotone fragment of modal logic.

Theorem 3.6 *[31]* HL-MC($\downarrow, \Box, \Diamond, \wedge, \vee$) *is* PSPACE*-complete.*

Proof. The PSPACE-hardness follows by a reduction from the validity problem for quantified Boolean formulas in conjunctive normal form QBF-VAL. An instance of QBF-VAL is for example $\exists a \forall b \exists c (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$. The model checking instance obtained from this has the Kripke model shown in Figure 4, initial state w_0, assignment g that maps every state variable to w_0, and a hybrid formula that is obtained as follows. Each $\forall x$ is replaced by $\Box\downarrow\mathsf{x}$, and each $\exists x$ by $\Diamond\downarrow\mathsf{x}$. Each appearance of a positive literal x is replaced by $\Diamond(\mathsf{x} \wedge t)$ and each negative literal $\neg x$ by $\Diamond(\mathsf{x} \wedge f)$. The propositional atoms used here are t and f, all other atoms are state variables. From the above formula we obtain $\Diamond\downarrow\mathsf{a}.\Box\downarrow\mathsf{b}.\Diamond\downarrow\mathsf{c}.(\Diamond(\mathsf{a} \wedge t) \vee \Diamond(\mathsf{b} \wedge t) \vee \Diamond(\mathsf{c} \wedge f)) \ldots$. The quantifiers in the quantified Boolean formula, that intuitively set propositional atoms to *true* or *false*, are simulated by the modal operators, that

assign a state variable to a state that satisfies either t or f. At the end, setting an atom x to *true* corresponds to assigning state variable x to the state that satisfies t. This setting is "stored" in the assignment g and can be checked by $\Diamond(\mathsf{x} \wedge t)$. □

4 Intuitionistic logic

Intuitionistic propositional logic goes back to Heyting and bases on Brouwer's idea of constructivism from the beginning of the 20th century. It can be seen as the part of propositional logic that goes without the use of the excluded middle $a \vee \neg a$. It lies intermediate between propositional and modal logic. Whereas its satisfiability is NP-complete as for propositional logic, its validity problem is PSPACE-complete [36] as for modal logic.

The language of intuitionistic logic is the set of all formulas of the form

$$\varphi \quad ::= \quad \bot \mid p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrowtriangle \varphi,$$

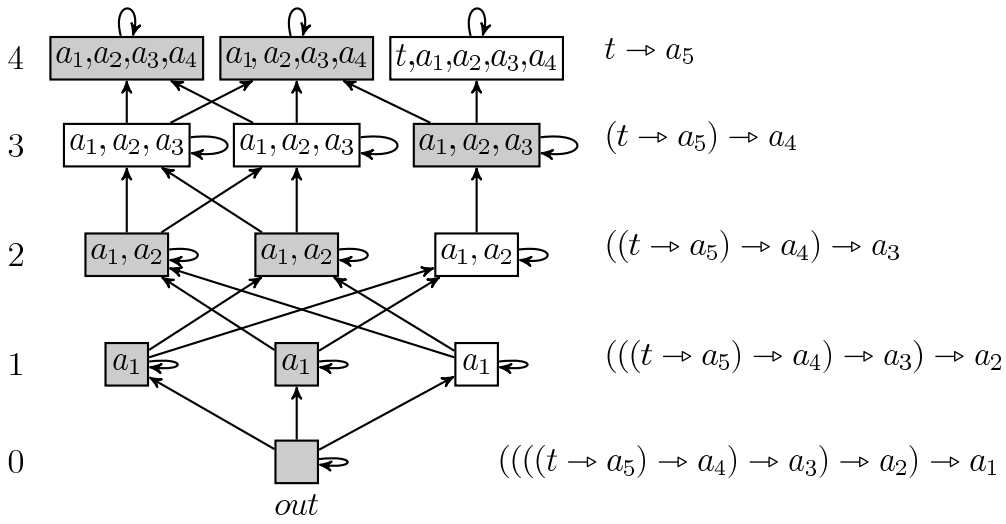
where p is a propositional atom. Notice that the negation $\neg$ and the exclusive or $\oplus$ do not appear in intuitionistic logic. Moreover, the semantics of $\rightarrowtriangle$ is different of the semantics of the implication $\rightarrow$ in classical propositional logic. The semantics is defined via Kripke models (W, R, ξ) that have the following properties. First, (W, R) is reflexive and transitive—therefore one uses often $\leq$ as relation symbol. Second, ξ is monotone in the way that if $w \in \xi(p)$ and $(w, v) \in R$ then $v \in \xi(p)$. We call such a Kripke model *intuitionistic*. Let $\mathcal{M} = (W, \leq, \xi)$ be an intuitionistic model with state $s \in W$. The semantics of intuitionistic logic extends the semantics of the $\{\wedge, \vee\}$-fragment of the basic language as follows.

$$\mathcal{M}, s \not\models \bot$$
$$\mathcal{M}, s \models \varphi \rightarrowtriangle \psi \quad \text{iff} \quad \forall w \geq s : \text{ if } \mathcal{M}, w \models \varphi \text{ then } \mathcal{M}, w \models \psi.$$

As $\neg\alpha \equiv \alpha \rightarrow \bot$ in classical propositional logic, negation in intuitionistic logic can be expressed as $\alpha \rightarrowtriangle \bot$ holds and has quite a different meaning. Other than $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$ in classical propositional logic, we have $\alpha \rightarrowtriangle \beta \not\equiv (\alpha \rightarrowtriangle \bot) \vee \beta$ in intuitionistic logic.

The model checking problem for modal logic is defined as follows.

Problem:	IPL-MC
Description:	the model checking problem for intuitionistic logic.
Input:	An intuitionistic formula ϕ, an intuitionistic Kripke model $\mathcal{M} = (W, \leq, \xi)$, and an initial state $w_0 \in W$.
Question:	Does $\mathcal{M}, w_0 \models \phi$ hold?



Intuitionistic Kripke model I_C

Figure 5: The intuitionistic Kripke model I_C constructed from the monotone Boolean circuit C in Figure 2. The pseudo-transitive edges are not drawn. The numbers on the left-hand side indicate the layers. Grey states satisfy the formula on the right-hand side, and white states do not satisfy it.

Gödel-Tarski translations map intuitionistic formulas to modal formulas in a way that preserves satisfaction and validity in the different logics. We take a translation from [40], that we call gt and that maps intuitionistic formulas on modal formulas as follows.

$$\begin{array}{llllll} gt(\bot) & := & \bot & gt(p) & := & p \quad \text{for atom } p \\ gt(\alpha \wedge \beta) & := & gt(\alpha) \wedge gt(\beta) & gt(\alpha \vee \beta) & := & gt(\alpha) \vee gt(\beta) \\ gt(\alpha \rightarrowtriangle \beta) & := & \Box(gt(\alpha) \rightarrow gt(\beta)) & & & \end{array}$$

For every intuitionistic Kripke model $\mathcal{M}$ with state w and every intuitionistic formula φ it holds that $\mathcal{M}, w \models \varphi$ if and only if $\mathcal{M}, w \models gt(\varphi)$. Therefore, P as upper bound for the model checking problem follows immediately from Fisher and Ladner [12].

Theorem 4.1 *[12]* IPL-MC *is in* P.

Completeness results were obtained in [23, 24, 25]. In general, the model checking problem for intuitionistic logic is P-complete. More detailed, P-hardness is reached already for intuitionistic formulas with $\rightarrowtriangle$ as only operator. Figure 5 shows an example for the intuitionistic Kripke model I_C and the formula constructed from the monotone circuit C in Figure 2. The states

of I_C are the nodes of C, and the edges of I_C include the reversed edges of C. An intuitionistic Kripke model must be transitive and reflexive. It is straightforward to make I_C reflexive. But since the reduction from the circuit to the intuitionistic Kripke model must be computable within logarithmic space, it is not possible to calculate the correct transitive edges. Instead, we take advantage that the circuit is layered and add "pseudo-transitive" edges that connect each node with every node that lies at least two layers above. The initial node *out* is in layer 0. The assignment assigns atom a_i to all nodes in all layers $\geq i$ and the atom t to all 0-input gates of the circuit. In layer 4, the formula $t \rightarrowtriangle a_5$ is satisfied in the states corresponding to input nodes with input 1. Formula $(t \rightarrowtriangle a_5) \rightarrowtriangle a_4$ is satisfied by all states in this layer, because a_4 is satisfied everywhere. In layer 3, a_4 is not satisfied, and therefore $(t \rightarrowtriangle a_5) \rightarrowtriangle a_4$ is satisfied in all states that don't satisfy $t \rightarrowtriangle a_5$. Since neither t nor a_5 is satisfied in layer 3, this depends on whether a state has a successor that doesn't satisfy $t \rightarrowtriangle a_5$—at this point the semantics of the intuitionistic implication $\rightarrowtriangle$ comes into play. These successors correspond to 0-input gates. This means, that in layer 3 exactly those states satisfy $(t \rightarrowtriangle a_5) \rightarrowtriangle a_4$, that correspond to $\wedge$-gates in the circuit that output 0. Accordingly, in this layer states that do not satisfy $(t \rightarrowtriangle a_5) \rightarrowtriangle a_4$ correspond to $\wedge$-gates in the circuit that output 1. Consequently, in layer 2 exactly those states satisfy $((t \rightarrowtriangle a_5) \rightarrowtriangle a_4) \rightarrowtriangle a_3$, that have a successor that does not satisfy $(t \rightarrowtriangle a_5) \rightarrowtriangle a_4$—i.e. exactly those states satisfy $((t \rightarrowtriangle a_5) \rightarrowtriangle a_4) \rightarrowtriangle a_3$, that correspond to $\vee$-gates in the circuit that output 1. This continues down to the state *out* in layer 0 that satisfy the corresponding formula if and only if the circuit outputs 1. The assignment to the atoms is used to "implement" the repeated negations that—due to the semantics of intuitionistic logic—simulate the layerwise alternation between $\wedge$-gates and $\vee$-gates of the circuit. Moreover, it arranges that we don't need to care about the pseudo-transitive edges.

Can we save the use of atoms? Intuitionistic formulas with a bounded number of atoms and the operators $\wedge$, $\rightarrowtriangle$, and $\bot$ have only a bounded number of equivalence classes [9]. This eventually causes the model checking problem to be in NC^1. Contrary, with only one atom and the operators $\rightarrowtriangle$ and $\vee$, there are infinitely many equivalence classes of intuitionistic formulas that form a Heyting algebra with one generator [26]. These structural properties are essential to show that the model checking problem gets AC^1-complete for intuitionistic formulas with one atom. This result is especially interesting because it is the first "natural" AC^1-complete problem. For formulas with two atoms, the problem gets P-complete and reaches its maximal hardness.

Theorem 4.2 *1. [23]* IPL-MC *is* P*-complete even for intuitionistic formulas with* $\rightarrow$ *as only operator.*

2. [25] IPL-MC *is* P*-complete even for intuitionistic formulas with only two atoms.*

3. [24] IPL-MC *is* AC^1*-complete for intuitionistic formulas with only one atom.*

The intuitionistic implication $\rightarrow$ is well-defined for modal logic—it can be seen as the operator $\Box(\cdot \rightarrow \cdot)$. In Theorem 3.2 we have seen that model checking is P-hard for the fragment of modal logic with $\Diamond$ and $\Box$ and without any Boolean operator or atom. The proof technique of Theorem 4.2(1) can also be applied to modal logic, where we do not have to deal with transitive frames. This allows to get P-hardness even without atoms.

The modal logic S4 is the restriction of modal logics to Kripke models (W, R, ξ) with a reflexive and transitive frame (W, R). S4 is called *modal companion* of intuitionistic logic, since there is a mapping—e.g. the Gödel-Tarski translation gt—that maps valid intuitionistic formulas to S4-valid modal formulas and non-valid intuitionistic formulas to non-S4-valid modal formulas. The above hardness results for intuitionistic logic also hold for S4. Other than for intuitionistic logic, it is open whether the S4 fragments with $\rightarrow$ as only operator and a bounded number of atoms have only a finite number of equivalence classes. For fragments with unrestricted operators and bounded number of atoms, it turns out that P-hardness is obtained for S4 with one atom less than for intuitionistic logic.

Theorem 4.3 *[25]*

1. ML-MC$(\Box(\cdot \rightarrow \cdot))$ *is* P*-complete for modal formulas that use constant symbol* $\bot$ *("false") and no atoms.*

2. ML-MC$(\Box, \Diamond, \wedge, \vee, \neg)$ *is* P*-complete for S4 models even for modal formulas with only one atom.*

More results that compare the complexities of model checking for different intuitionistic logics and their modal companions can be found in [25]. It seems that if a modal logic has a P-complete model checking problem, then also its fragment with the intuitionistic implication as only operator has a P-complete model checking problem. There are only differences between the fragments in the number of atoms needed to show this hardness.

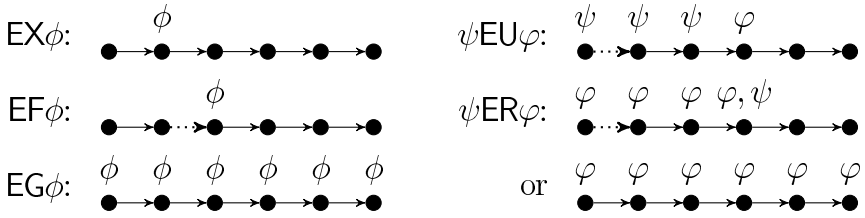


Figure 6: Semantics of existential CTL operators.

5 CTL

Computation Tree Logic CTL is an extension of modal logic. If one considers a Kripke model as a set of states of a computer program with the possible transitions between these states within one step of time, a modal logic formula can only express properties of a run of the program for some fixed number of steps. In CTL, one can express properties of the possibly infinite run of the program. It can be dated back up to Prior [29, 30]. However, Pnueli built the basic notion and the framework around it [27], and afterwards Clarke and Emerson refined it to what is used today [8]. As we will see later, CTL is a strict subset of the more general full branching time logic CTL*.

The set of all CTL formulas extends the basic language from Section 2 by the unary operators EX, AX, EG, AG, EF, AF and the binary operators EU, AU, ER, AR, that can be seen as combined from the path quantifiers E ("for some path") and A ("for all paths") as for CTL* and the LTL operators X ("next"), F ("future"), and G ("globally"), U ("until") and R ("release"). The semantics of these temporal operators is defined on states of total Kripke models.

The semantics of CTL is defined on states, but also uses paths. This makes model checking easier for CTL than for the path related temporal logics LTL and CTL*. Let $\mathcal{M} = (W, R, \xi)$ be a total Kripke model. Remind that $\Pi(w)$ is the set of infinite paths starting in w through $\mathcal{M}$. The semantics of CTL extends the semantics of the basic language as follows.

$$
\begin{array}{lll}
\mathcal{M}, w \models \mathsf{EX}\varphi & \text{iff} & \exists \pi \in \Pi(w) : \ \mathcal{M}, \pi[2] \models \varphi, \\
\mathcal{M}, w \models \mathsf{AX}\varphi & \text{iff} & \forall \pi \in \Pi(w) : \ \mathcal{M}, \pi[2] \models \varphi, \\
\mathcal{M}, w \models \mathsf{EF}\varphi & \text{iff} & \exists \pi \in \Pi(w) \ \exists k \geq 1 : \ \mathcal{M}, \pi[k] \models \varphi, \\
\mathcal{M}, w \models \mathsf{AF}\varphi & \text{iff} & \forall \pi \in \Pi(w) \ \exists k \geq 1 : \ \mathcal{M}, \pi[k] \models \varphi, \\
\mathcal{M}, w \models \mathsf{EG}\varphi & \text{iff} & \exists \pi \in \Pi(w) \ \forall k \geq 1 : \ \mathcal{M}, \pi[k] \models \varphi, \\
\mathcal{M}, w \models \mathsf{AG}\varphi & \text{iff} & \forall \pi \in \Pi(w) \ \forall k \geq 1 : \ \mathcal{M}, \pi[k] \models \varphi,
\end{array}
$$

$$
\begin{aligned}
\mathcal{M}, w \models \psi \mathsf{EU} \varphi \quad \text{iff} \quad & \exists \pi \in \Pi(w)\ \exists k \geq 1 : \\
& \mathcal{M}, \pi[k] \models \varphi \text{ and } \forall i < k : \mathcal{M}, \pi[i] \models \psi, \\
\mathcal{M}, w \models \psi \mathsf{AU} \varphi \quad \text{iff} \quad & \forall \pi \in \Pi(w)\ \exists k \geq 1 : \\
& \mathcal{M}, \pi[k] \models \varphi \text{ and } \forall i < k : \mathcal{M}, \pi[i] \models \psi, \\
\mathcal{M}, w \models \psi \mathsf{ER} \varphi \quad \text{iff} \quad & \exists \pi \in \Pi(w)\ \forall k \geq 1 : \\
& \mathcal{M}, \pi[k] \models \varphi \text{ or } \exists i \leq k : \mathcal{M}, \pi[i] \models \psi, \\
\mathcal{M}, w \models \psi \mathsf{AR} \varphi \quad \text{iff} \quad & \forall \pi \in \Pi(w)\ \forall k \geq 1 : \\
& \mathcal{M}, \pi[k] \models \varphi \text{ or } \exists i \leq k : \mathcal{M}, \pi[i] \models \psi.
\end{aligned}
$$

We have the equivalences between $\mathsf{EX}\phi \equiv \neg\mathsf{AX}\neg\phi, \mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$, $\mathsf{EF}\phi \equiv \neg\mathsf{AG}\neg\phi$, and similarly EU/AU is dual to AR/ER. Moreover, the operators $\mathsf{EX}, \mathsf{EG}, \mathsf{EU}$ are a minimal set of CTL operators that together with the Boolean operators suffice to express any from the left overs [17].

Problem:	CTL-MC(T)
Description:	the model checking problem for CTL.
Input:	a CTL formula ϕ with operators in $T \subseteq \{\mathsf{EX}, \mathsf{AX}, \mathsf{EG}, \mathsf{AG}, \mathsf{EF}, \mathsf{AF}, \mathsf{EU}, \mathsf{AU}, \mathsf{ER}, \mathsf{AR}, \wedge, \vee, \neg, \oplus\}$, a total Kripke model $\mathcal{M} = (W, R, \xi)$, and an initial state w_0.
Question:	Does $\mathcal{M}, w_0 \models \phi$ hold?

Clarke, Emerson, and Sistla [6] showed that model checking for CTL is in P. The algorithm is an extension of the model checking algorithm for modal logic. The marking of the states with $\mathsf{EX}\alpha$ subformulas works as for $\Diamond$ in modal logic. The marking with $\mathsf{EG}\alpha$ subformulas is a little more involved. At first, every state that has mark α an lies on a cycle of state with mark α is marked with $\mathsf{EG}\alpha$. After this, all states marked with α that have a neighbour marked with $\mathsf{EG}\alpha$ are marked with $\mathsf{EG}\alpha$, too. The marking with $\alpha\mathsf{EU}\beta$ subformulas also works inductively. At first, every state that has mark β is marked with $\alpha\mathsf{EU}\beta$. After this, repeatedly all states with mark α that have a neighbour with mark $\alpha\mathsf{EU}\beta$ are marked with $\alpha\mathsf{EU}\beta$. The remaining operators can be expressed using the above operators, and accordingly the marking can be made.

Theorem 5.1 *[6]* CTL-MC($\mathsf{EX}, \ldots, \mathsf{AR}, \wedge, \vee, \neg, \oplus$) *is in* P.

The proof idea for the P-hardness is similar to that of modal logic (see Theorem 3.2), where the $\Diamond$ and $\Box$ operators can be replaced by EX and AX (see [6, 32]). The only difference to modal logic is that one cannot get rid of

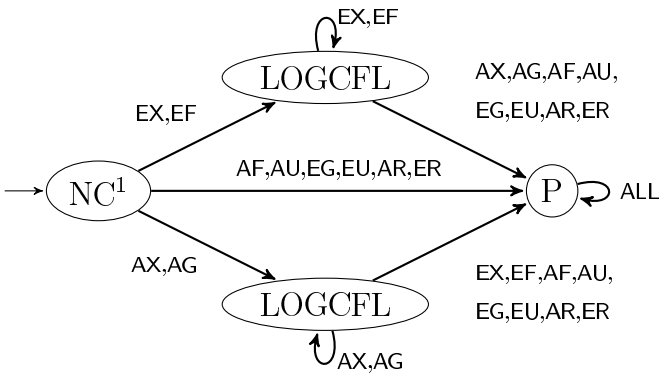


Figure 7: The completeness results of model checking for CTL with the Boolean operators $\wedge$ and $\vee$ (Theorem 5.2).

the only atom used, since the Kripke models for CTL are total. Thus we have P-hardness for the $\{\mathsf{EX}, \mathsf{AX}\}$-fragment of CTL without Boolean operators. This hardness does not necessarily apply for other pairs of dual operators. The only solved case is for the $\{\mathsf{EU}\}$-fragment (see Theorem 5.5), where one needs only one temporal and no Boolean operator.

How temporal operators affect the complexity of model checking, was investigated in [2] for monotone formulas. It came out that some conjunctive and some disjunctive property of the temporal operators is necessary to obtain P-hardness for a monotone CTL fragment. The conjunctive property appears for A, G and U, and the disjunctive property appears for E, F and U. Notice that U has both properties, whereas X has none of these properties. Thus the CTL operator EG has both properties, because E is disjunctive and G is conjunctive. Also, both EU and AU have both properties, whereas EX is only disjunctive. If we take both operators EX and AX together, then they combine both properties.

Theorem 5.2 *[2]*

1. CTL-MC$(S, \wedge, \vee)$ *is* P*-complete if and only if* S *is a set of CTL operators that has the disjunctive and the conjunctive property.*

2. CTL-MC$(S, \wedge, \vee)$ *is* LOGCFL*-complete if* S *is a set of CTL operators that are either all disjunctive or all conjunctive.*

Below, we will extend the above result by a complete characterization of the complexity for all fragments that have exactly one of the basic temporal operators EX, EG, and EU. CTL fragments with EX as only temporal operator are considered in Theorem 5.3. They are essentially the same as

modal fragments with $\Diamond$ as only modal operator. In Theorem 5.4 we consider fragments with EG. This operator deals essentially with paths, other than EX oder the modal operators. We sketch the proof idea of the P-hardness for the $\{\mathsf{EG}, \wedge, \vee\}$-fragment of CTL from [2], that is a typical construction in the proof of Theorem 5.2. An appropriate modification can be used to show P-hardness for the $\{\mathsf{EG}, \oplus\}$-fragment. It turns out, that formulas with EG and exactly one of the Boolean operators $\wedge$, $\vee$, and $\neg$ cannot search through trees in the Kripke models but only through paths. Therefore, the model checking problem for these fragments is in NL. Theorem 5.5 shows that the $\{\mathsf{EU}\}$-fragment is already P-hard, i.e. Boolean operators are not necessary in this case.

It is interesting to notice that the EX-fragments characterize three complexity classes NL, LOGCFL, and P, whereas the EG-fragments distribute over two complexity classes, and the EU-fragments fall into one complexity class only. An overview of these results is shown in Figure 9.

Theorem 5.3 CTL-MC(EX, B) *is*

1. P-*complete for* $B \supseteq \{\neg\}$ *or* $B \supseteq \{\oplus\}$,
2. LOGCFL-*complete for* $B = \{\wedge, \vee\}$ *or* $B = \{\wedge\}$ *[2], and*
3. NL-*complete for* $B \subseteq \{\vee\}$.

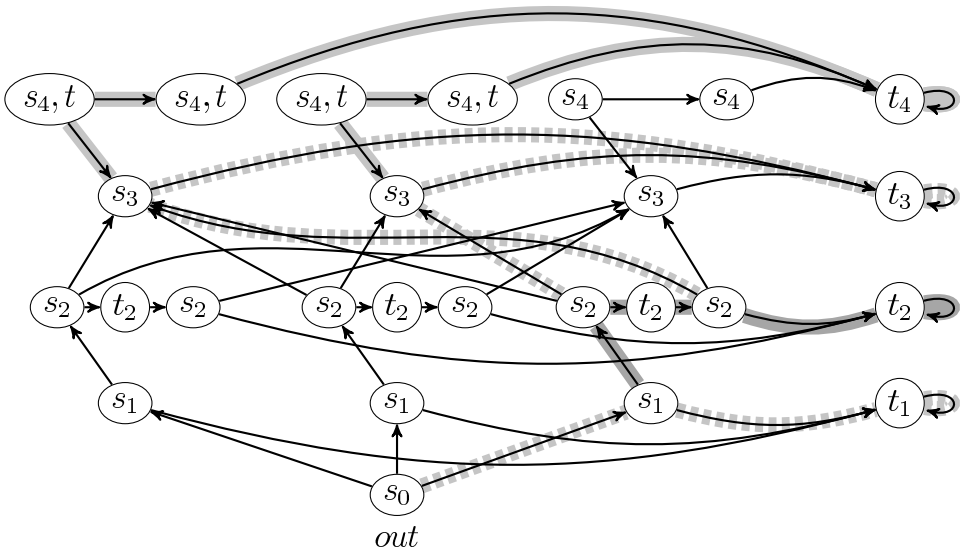
Proof. Since EX and AX can essentially be seen as the modal operators $\Diamond$ and $\Box$, the results follow directly from Theorem 3.2. □

Thus, EX-fragments reach their maximal hardness only with $\neg$. This is due to the fact that with $\wedge$ and $\vee$ one can "check" only polynomial size trees in a Kripke model. With EG-fragments instead, exponential size trees can be "checked" with monotone formulas.

Theorem 5.4 CTL-MC(EG, B) *is*

1. P-*complete for* $B \supseteq \{\wedge, \vee\}$ *[2] or* $B \supseteq \{\oplus\}$,
2. NL-*complete for* $B \subsetneq \{\wedge, \vee\}$ *or* $B \subseteq \{\neg\}$.

Proof. 1. The complete proof for CTL-MC($\mathsf{EG}, \wedge, \vee$) can be found in [2]. We give a rough sketch of the construction for the reduction from the monotone Boolean circuit evaluation problem to CTL-MC($\mathsf{EG}, \wedge, \vee$). For simplicity, we assume that the $\wedge$-gates have indegree 2. At first, from a circuit C we construct the Kripke model $\hat{K}_C$ as in the proof of Theorem 3.3 (see also Figure 2). Next we switch the inputs to $\wedge$-nodes in series by putting an edge



Kripke model $\hat{C}_{\mathsf{EG}}$

Figure 8: Kripke model $\hat{C}_{\mathsf{EG}}$ (proof of Theorem 5.4(1)) constructed from the monotone circuit in Figure 2. The grey shades indicate the tree consisting of 6 paths that show how the formula ϕ_0 is satisfied.

from the left to the right input of every $\wedge$-node and by removing the edge from a $\wedge$-node to its right input. Intermediate between both the input nodes we put a node with t_i for layer i. Third we add a node marked with t_i for every layer i of the graph. This node has a loop and has incoming edges from every node of its layer, if the layer consists of inputs to $\vee$-gates, or from every right input to a $\wedge$-gate, if the layer consists of inputs to $\wedge$-gates. The assignment function assigns t_i to every node t_i, and s_i to all other nodes in layer i. The 1-inputs are assigned with t. This yields the Kripke model $\hat{C}_{\mathsf{EG}}$—see Figure 8 for an example. Finally define recursively $(\phi_i)_{0 \le i \le \ell}$ by

$$\phi_i := \begin{cases} \mathsf{EG}(t \vee t_{\ell+1}), & \text{if } i = \ell, \\ \mathsf{EG}(s_i \vee t_{i+1} \vee (s_{i+1} \wedge \phi_{i+1})), & \text{if } i < \ell. \end{cases}$$

Then it holds that circuit C outputs 1 if and only if $C_{\mathsf{EG}}, \mathit{out} \models \phi_0$. The proof idea is as follows. At first, consider $\phi_0 = \mathsf{EG}(s_0 \vee t_1 \vee (s_1 \wedge \phi_1))$. The path through the model that is bounded by the E must go from *out* to t_1 and intermediately passes a node on which s_1 holds and which also satisfies ϕ_1. This $\phi_1 = \mathsf{EG}(s_1 \vee t_2 \vee (s_2 \wedge \phi_2))$ forces a path from a s_1 node to the right most t_2 node, that passes the s_2–t_2–s_2 node sequence. Remind that the s_1

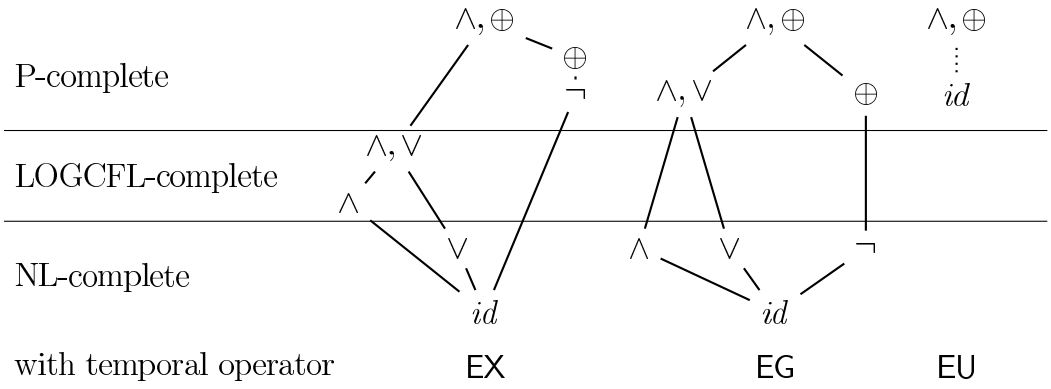


Figure 9: Complete characterizations of the complexity of model checking for fragments of CTL with one temporal operator EX, EG, resp. EU.

node origins from an $\wedge$-gate and that the two s_2 nodes are both inputs to this $\wedge$-gate. On both s_2 nodes ϕ_2 must be satisfied. This is the way in which an $\wedge$-gate is simulated. An $\vee$-gate is simulated just by forcing the path to go immediately to a node in the next layer and to the t_i node of this layer afterwards. It turns out that the tree of paths that witnesses the satisfaction of ϕ_0 corresponds to the substructure of the monotone circuit that witnesses that its output equals 1.

Notice that the $\vee$s we use are essentially used like $\oplus$. The P-hardness of CTL-MC(EG, $\oplus$) is therefore obtained by an appropriate modification of this construction, that makes it possible to avoid the use of the $\wedge$.

2. The NL-completeness results follow with usual arguments. □

Eventually we consider EU. The model checking problem is P-complete for the {EU}-fragment without any further Boolean operators. One of the reasons for this hardness is that EU is a two-place operator and cannot be expressed by any Boolean combination of the other temporal operators. Therefore, there may be some "Boolean power" hidden in EU.

Theorem 5.5 CTL-MC(EU) *is* P*-complete.*

Proof. In order to show P-hardness, we use a similar construction of a Kripke model from a monotone Boolean circuit as in the proof of Theorem 5.4(1). Let $\hat{C}_{\mathsf{EG}}$ be the Kripke model constructed from a circuit C as defined above. In order to obtain C_{EU}, change the assignment to every right input of an $\wedge$-gate from s_i to s'_i, and remove the intermediate nodes. Let u_i be a node corresponding to the left input of an $\wedge$-gate, and let u'_i be the node corresponding to the right input of the same $\wedge$-gate, that is now the "right" neighbour of u_i. The u_i satisfies s_i, and u'_i satisfies s'_i. Let α be not satisfied neither in u_i nor

in u'_i. Then $u_i \models s'_i \mathsf{EU}\,(s_i \mathsf{EU} \alpha)$ iff u_i has an "upper" neighbour that satisfies α. And $u'_i \models s'_i \mathsf{EU}\,(s_i \mathsf{EU} \alpha)$ iff u'_i has an "upper" neighbour that satisfies α. Thus $u_i \models (s'_i \mathsf{EU}\,(s_i \mathsf{EU} \alpha)) \mathsf{EU} t_i$ iff both u_i and u'_i have "upper" neighbours that satisfy α. This is the way in which an $\wedge$-gate can be simulated using EU as only operator and leads to the following recursive definition of $(\phi_i)_{0 \le i \le \ell}$.

$$\phi_i := \begin{cases} t \,\mathsf{EU}\, t_{\ell+1}, & \text{if } i = \ell, \\ s_i \,\mathsf{EU}\, \phi_{i+1}, & \text{if layer } i \text{ consists of or-gates,} \\ s_i \,\mathsf{EU}\, ((s'_{i+1} \,\mathsf{EU}\, (s_{i+1} \,\mathsf{EU}\, \phi_{i+1})) \,\mathsf{EU}\, t_{i+1}), & \text{if layer } i \text{ cons. of and-gates.} \end{cases}$$

Then it holds that circuit C outputs 1 if and only if $C_{\mathsf{EU}}, \mathit{out} \models \phi_0$. □

A complete characterization for fragments e.g. with EF as only temporal operator or for many combinations with two temporal operators is open.

6 LTL

Linear Temporal Logic (LTL) has been proposed by [27] as a formalism to specify properties of parallel programs and concurrent systems, as well as to reason about their behaviour. The language of LTL extends the basic propositional language with the unary operators X ("next"), F ("future"), and G ("globally"), and the binary operators U ("until") and R ("release"). The semantics of these temporal operators is defined on total Kripke models $\mathcal{M} = (W, R, \xi)$ and infinite paths π through it, as follows.

$$\begin{array}{lll} \mathcal{M}, \pi \models \mathsf{X}\varphi & \text{iff} & \mathcal{M}, \pi^2 \models \varphi \\ \mathcal{M}, \pi \models \mathsf{F}\varphi & \text{iff} & \exists j \ge 1 : \ \mathcal{M}, \pi^j \models \varphi \\ \mathcal{M}, \pi \models \mathsf{G}\varphi & \text{iff} & \forall j \ge 1 : \ \mathcal{M}, \pi^j \models \varphi \\ \mathcal{M}, \pi \models \varphi \mathsf{U} \psi & \text{iff} & \exists \ell \ge 1 : \ \mathcal{M}, \pi^\ell \models \psi \text{ and } \forall j, 1 \le j < \ell : \mathcal{M}, \pi^j \models \varphi \\ \mathcal{M}, \pi \models \varphi \mathsf{R} \psi & \text{iff} & \forall \ell \ge 1 : \ \mathcal{M}, \pi^\ell \models \psi \ \text{ or } \ \exists j, 1 \le j < \ell : \mathcal{M}, \pi^j \models \varphi \end{array}$$

Notice the dualities $\mathsf{F}\varphi \equiv \neg\mathsf{G}\neg\varphi$ and $\varphi \mathsf{U} \psi \equiv \neg(\neg\varphi \mathsf{R} \neg\psi)$. Moreover, $\mathsf{F}\varphi \equiv \top \mathsf{U} \varphi$. Therefore, the temporal operators X and U together with the Boolean operator $\neg$ suffice to express all temporal operators used here.

The model checking problem for LTL asks, whether there exists some infinite path through a given Kripke model that satisfies the given formula.

This problem statement is also called *existential* model checking. There is also a *universal* model checking problem considered in the literature, where the question is "Does *for all* paths π through $\mathcal{M}$ that start in w hold $\mathcal{M}, \pi \models \phi$?". The results are dual, and therefore we only talk about existential model checking in this section.

Problem:	LTL-MC(T)
Description:	the model checking problem for LTL.
Input:	an LTL formula ϕ using only operators in $T \subseteq \{\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}, \wedge, \vee, \neg, \oplus\}$, a total Kripke model $\mathcal{M} = (W, R, \xi)$, and an initial state w_0.
Question:	Is there an infinite path π through $\mathcal{M}$ that starts in $\pi[1] = w_0$, such that $\mathcal{M}, \pi \models \phi$ holds?

Sistla and Clarke [35] have shown that model checking for LTL is in PSPACE. Other than with the CTL model checking algorithm, that considers for every state and every subformula, whether the subformula is satisfied in that state, for LTL we additionally would have to take paths into account. This makes it impossible to use a dynamic programming approach as for CTL and modal logic. Instead, one copies every state s of the Kripke for all non-contradictory assignments of truth values to the subformulas of φ that extend the assignment to the atoms of the state. For example, if $s \in \xi(a)$, then a is assigned to *true*. Consequently, $\neg a$ must be assigned to *false*. If α and β are assigned to false, then $\alpha \mathsf{U} \beta$ must also be assigned to false, and so on. The edges of this graph are according to the edges of the Kripke model, but must additionally respect a consistency of the assignments to the subformulas. For example, if $\mathsf{X}\alpha$ is assigned to true, all edges go to states in which α is assigned to true. The size of the graph is exponential in the size of the formula and the Kripke model. In [35] the model checking algorithm uses this graph as follows. For simplicity, we only consider formulas with temporal operators X and U. In order to check whether φ is satisfied for some path π that starts in s, the algorithm guesses a copy of s in which φ is assigned to true. It then guesses a path to some node t and a loop that again leads to t, such that in this loop for every subformula $\alpha \mathsf{U} \beta$ of φ some state appears that assigns true to β. Because one could loop forever now, this is an infinite path. Such a path—that corresponds to an infinite path through the Kripke model—exists if and only if φ is satisfied for some path π through the Kripke model that starts in s. Since the size of the graph is exponential but is uniquely described by the Kripke model and the formula, this search can be (nondeterministically) done in polynomial space. Later on, Vardi and Wolper [39] developed a connection with the theory of automata over infinite words.

Theorem 6.1 *[35]* LTL-MC($\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}, \wedge, \vee, \neg, \oplus$) *is in* PSPACE.

Sistla and Clarke [35] also considered the hardness of fragments of LTL w.r.t. the temporal operators and showed that LTL-MC($\mathsf{U}, \wedge, \oplus$) is PSPACE-hard. This shows that the until operator makes LTL formulas hard to check,

whereas the full expressive power of LTL is not reached only with U, but with X and U. Markey [20] improved this result for the fragment where the until operators do not appear within the scope of a negation. Whereas $\mathsf{G}\alpha \equiv \neg(\top \mathsf{U} \neg\alpha)$, in this fragment the G operator cannot be expressed. In [1] it is shown that negations are not necessary for PSPACE-hardness in the fragment with U. Moreover, model checking is intractable for formulas that have U as only temporal operator and have no Boolean operators at all.

Theorem 6.2 *[1]*

1. LTL-MC($\mathsf{U}, \wedge, \vee$) *and* LTL-MC($\mathsf{R}, \wedge, \vee$) *are* PSPACE-*complete (cf. [35, 20]).*
2. LTL-MC(U) *and* LTL-MC(R) *are* NP-*hard.*

The PSPACE-hardness proof of LTL-MC($\mathsf{U}, \wedge, \vee$) can be done by a reduction from a tiling problem. As an example how this works in general, we consider model checking for the $\{\mathsf{G}, \mathsf{X}, \wedge, \vee\}$-fragment of LTL. The proof for the $\{\mathsf{U}, \wedge, \vee\}$-fragment is similar but technically more involved.

Theorem 6.3 *[20, 1]* LTL-MC($\mathsf{G}, \mathsf{X}, \wedge, \vee$) *is* PSPACE-*complete.*

Proof. Due to Theorem 6.1 it suffices to show PSPACE-hardness. We give a reduction from a PSPACE-hard tiling problem as in [20]. A *domino type* is a 4-tuple $(west, north, east, south)$ of colours. For a finite set T of domino types, a tiling of a rectangle with breadth b and height h is an arrangement that puts dominoes of the given type on every point of the rectangle such that horizontally or vertically neighboured dominos have the same color at the adjoining sides. Formally, this can be expressed as a function $f : \{1, 2, \ldots, b\} \times \{1, 2, \ldots, h\} \to T$ that fulfills the following properties. Let $f(i, j).south$ denote the *south* color of domino type $f(i, j)$ etc.

1. $f(i, j).east = f(i + 1, j).west$, for $i = 1, 2, \ldots, b - 1$ and $j = 1, 2, \ldots, h$
2. $f(i, j).north = f(i, j{+}1).south$, for $i = 1, 2, \ldots, b$ and $j = 1, 2, \ldots, h{-}1$.

The PSPACE-complete *corridor tiling problem* that we will use is defined as follows. Given a finite set T of domino types, two types $q, r \in T$ and a string 1^b of b ones, does the corridor of breadth b has a tiling with domino $q = f(1, 1)$ in the left lower corner and domino $r = f(b, h)$ in the right upper corner?

Given an instance $\tau = (T, q, r, 1^b)$ of the tiling problem, the construction of a LTL-MC(G, X)M instance (K, w_0, φ) works as follows. Let $T = \{t_0, t_1, \ldots, t_k\}$ be the finite set of domino types. The frame of the Kripke

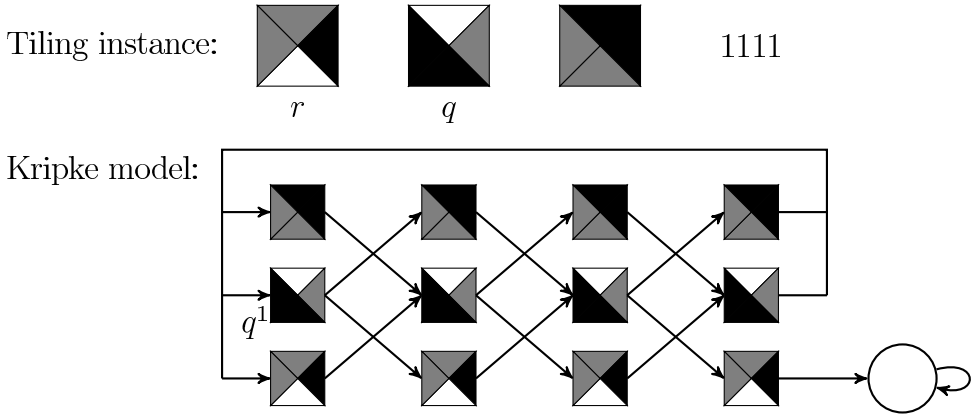


Figure 10: A corridor tiling instance and a sketch of the Kripke model constructed from it.

model is a graph such that paths through this graph represent sequences of horizontally correctly tiled rows—the vertical correctness will be checked by the LTL formula. The graph has nodes $\{t_i^j \mid 0 \leq i \leq k, 1 \leq j \leq b\}$ where each t_i^j represents a copy of domino type t_i in column j of a row. The edges in the graph are between two domino types in neighboured columns whose east resp. west colour are equal (according to property 2 above), i.e. $\{(t_l^j, t_r^{j+1}) \mid 1 \leq l < b, t_l.east = t_r.west\}$. The nodes at the right end of a row have edges to all nodes at the left end of a row, if the node's type is not r. If it is r, then the only edge from it goes to a sink node, that only has an edge to itself. Figure 10 shows an example.

Let $C = \{c_1, \ldots, c_\ell\}$ be the colours that appear in T. The atoms used in our Kripke model represent the colours at the different sides of the domino types and are $\{n\text{-}c_i, e\text{-}c_i, s\text{-}c_i, w\text{-}c_i \mid 1 \leq i \leq \ell\}$. The assignment ξ maps state t_i^j having *north* colour c_{north} and so on to $\{n\text{-}c_{north}, e\text{-}c_{east}, s\text{-}c_{south}, w\text{-}c_{west}\}$. The sink node does not satisfy any atom. Now we have constructed Kripke model K_τ from a tiling problem τ given by $T, q, r, 1^b$.

The formula must check that in every state of the path a *north* colour exists—this guarantees that the sink state is not reached—and that it equals the *south* colour in the state reached b steps later—this means the domino in the row above fits.

$$\varphi_\tau = \mathsf{G} \bigvee_{c_i \in C} (n\text{-}c_i \wedge \mathsf{X}^b s\text{-}c_i)$$

It is not hard to see that $\tau = (T, q, r, 1^b)$ is in the corridor tiling problem if and only if there exists an infinite path π through K_τ that starts in q^1 such that $K_\tau, \pi \models \varphi_\tau$. □

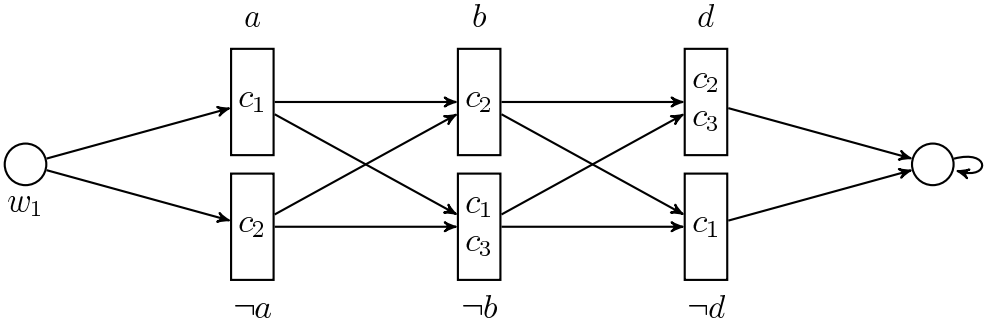


Figure 11: A Kripke model from $\alpha = (a \vee \neg b \vee \neg d) \wedge (\neg a \vee b \vee d) \wedge (\neg b \vee d)$ for the reduction from 3SAT to CTL-MC($\mathsf{F}, \wedge$) and to CTL-MC($\mathsf{X}, \wedge, \vee$).

It is interesting to notice that nesting of the G operator is not necessary in order to obtain the maximal computational hardness. This is essentially different from CTL, where nesting of EG operators was necessary in order to "construct" trees. Similar as for CTL it appears that dealing the G operator for the F operator decreases the complexity.

Theorem 6.4 *[35]* LTL-MC($\mathsf{F}, \mathsf{X}, \wedge, \vee$) *is* NP*-complete.*

The NP upper bound relies on the fact that $\{\mathsf{F}, \mathsf{X}, \wedge, \vee\}$-formulas can deal only with a polynomially long prefix of a path through a Kripke model. As examples for typical NP-hardness proofs, we consider the smallest NP-hard fragments.

Theorem 6.5 *[35, 20, 1] The following problems are* NP*-hard:*
LTL-MC($\mathsf{F}, \wedge$)*,* LTL-MC($\mathsf{X}, \wedge, \vee$)*, and* LTL-MC($\mathsf{G}, \mathsf{X}, \vee$)*.*

Proof. We start with the NP-hardness of LTL-MC($\mathsf{F}, \wedge$) [35]. The reduction is from the NP-complete problem CNF-SAT that asks whether a propositional formula given in conjunctive normal form is satisfiable. Let α be such a formula with atoms $x_1, \ldots, x_n$. The Kripke model K_α constructed from this formula has an initial state w, a sink state, and in between one state for each literal $x_1, \neg x_1, \ldots, x_n, \neg x_n$ that are connected in a way that every infinite path starting at w corresponds to exactly one assignment to the atoms in α—see Figure 11 for an example. Let $c_1, \ldots, c_m$ be the clauses of α. The assignment function assigns c_i to every state that corresponds to a literal that appears in clause c_i. This means, that a path through the model corresponds to a satisfying assignment if and only if it passes states in which eventually all c_is are satisfied. The satisfiability of α is the expressed by the LTL formula $\varphi = \mathsf{F}c_1 \wedge \mathsf{F}c_2 \wedge \cdots \wedge \mathsf{F}c_m$.

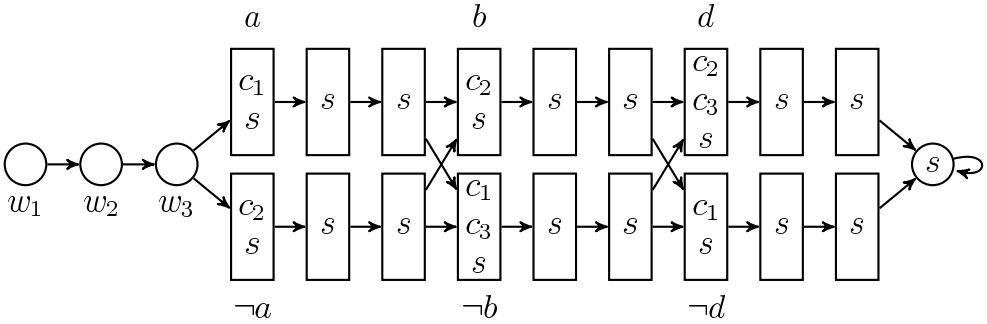


Figure 12: A Kripke model from the CNF formula $\alpha = (a \vee \neg b \vee \neg d) \wedge (\neg a \vee b \vee d) \wedge (\neg b \vee d)$ for the reduction from 3SAT to CTL-MC($\mathsf{G}, \mathsf{X}, \vee$).

In the same way one can prove the NP-hardness of LTL-MC($\mathsf{X}, \wedge, \vee$) [10, 20]. The reduction from CNF-SAT uses the same Kripke model as above and the LTL formula $(\mathsf{X}c_1 \vee \mathsf{XX}c_1 \vee \ldots \vee \mathsf{X}^n c_1) \wedge \ldots \wedge (\mathsf{X}c_m \vee \mathsf{XX}c_m \vee \ldots \vee \mathsf{X}^n c_m)$, for short $\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} \mathsf{X}^j c_i$.

One can get rid of the $\wedge$s by adding a G. For this, we modify the Kripke model from the above proof by making as follows. Each state (excepted the sink state) is replaced by a chain of as many copies as the CNF-SAT formulas has clauses. The assignment to the new copies is s, if the state corresponds to the assignment to the CNF-SAT formula, or empty otherwise. Figure 12 shows an example. Notice that $\psi_1 = \mathsf{X}^3 c_1 \vee \mathsf{X}^6 c_1 \vee \mathsf{X}^9 c_1$ can only be satisfied in state w_1. Accordingly, $\psi_2 = \mathsf{X}^2 c_2 \vee \mathsf{X}^5 c_2 \vee \mathsf{X}^8 c_2$ can only be satisfied in state w_2, and $\psi_3 = \bigvee_{i=1,4,7} \mathsf{X}^i c_3$ can only be satisfied in state w_3. Since s is satisfied in the remaining states, $\mathsf{G}(\psi_1 \vee \psi_2 \vee \psi_3 \vee s)$ is satisfied on exactly the paths through the Kripke model that correspond to satisfying assignments to the CNF-SAT formula. This is the idea for the proof that LTL-MC($\mathsf{G}, \mathsf{X}, \vee$) is NP-hard [1]. □

Whereas LTL-MC($\mathsf{F}, \wedge$) and LTL-MC($\mathsf{X}, \wedge, \vee$) are in NP by Theorem 6.4, it is open, whether LTL-MC($\mathsf{G}, \mathsf{X}, \vee$) is in NP.

In general, there is no easy rule for the hardness of model checking LTL fragments. The hardest cases—PSPACE-completeness—are reached with both G and X, or with U (resp. R). There are many NP-hard cases with G or with U (resp.R), where an upper bound below PSPACE is open. All cases that are not NP-hard have an NL-complete model checking problem. An overview over some results is given in Figure 13.

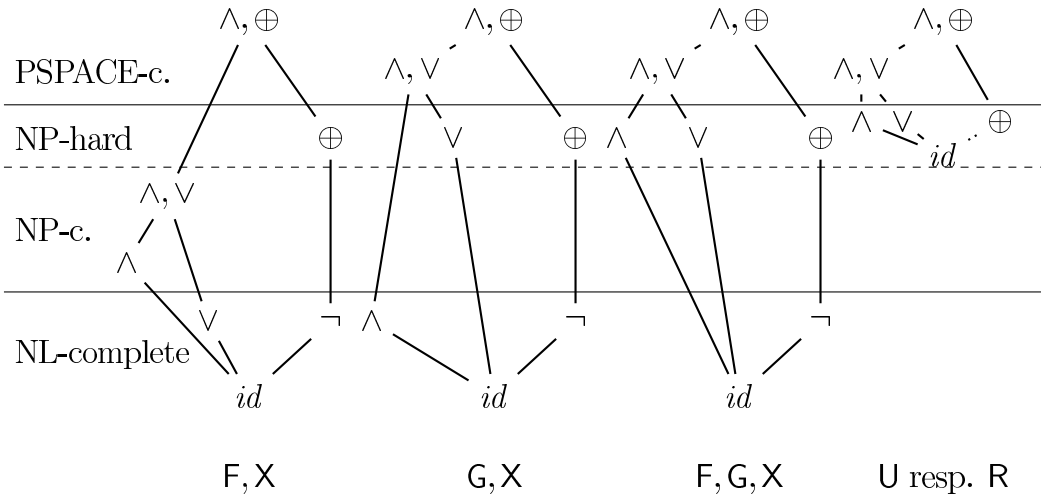


Figure 13: Overview over the complexity of model checking for LTL fragments with F and X, or G and X, or F, G, and X, or U resp. R (see [35, 20, 1, 15]).

7 CTL*

The full branching time logic CTL* is a more general version of CTL and LTL in the sense that one may require properties holding over several computation paths. Therefore the syntax extends the LTL language by the unary operators E and A. Let $\mathcal{M}$ be a Kripke model, w be a state in $\mathcal{M}$ and π be an infinite path through $\mathcal{M}$. The semantics of LTL is extended as follows. The E and A operators switch from state to path semantics.

$$\begin{aligned}
\mathcal{M}, w &\models \mathsf{E}\varphi \quad \text{iff} \quad \exists \pi \in \Pi(w) : \ \mathcal{M}, \pi \models \varphi, \\
\mathcal{M}, w &\models \mathsf{A}\varphi \quad \text{iff} \quad \forall \pi \in \Pi(w) : \ \mathcal{M}, \pi \models \varphi, \\
\mathcal{M}, \pi &\models \mathsf{E}\varphi \quad \text{iff} \quad \exists \pi' \in \Pi(\pi[1]) : \ \mathcal{M}, \pi' \models \varphi, \\
\mathcal{M}, \pi &\models \mathsf{A}\varphi \quad \text{iff} \quad \forall \pi' \in \Pi(\pi[1]) : \ \mathcal{M}, \pi' \models \varphi.
\end{aligned}$$

The model checking problem for CTL* asks, whether a formula is satisfied in a given state of a Kripke model.

Problem:	CTL*-MC(T)
Description:	the model checking problem for CTL*.
Input:	a CTL* formula ϕ using only operators in $T \subseteq \{\mathsf{E}, \mathsf{A}, \mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}, \wedge, \vee, \neg, \oplus\}$, a total Kripke model $\mathcal{M} = (W, R, \xi)$, and an initial state w_0.
Question:	Does $\mathcal{M}, w_0 \models \phi$ hold?

The model checking task for CTL* is PSPACE-complete which has been shown by Clarke et al in 1986 [6]. Given a formula ϕ, one translates the formula into a form such that it does not contain any A's. Then the main idea is to recursively evaluate subformulas $\mathsf{E}\psi$ (s.t. ψ contains no E) in ϕ with the LTL algorithm (see Theorem 6.1) to check in which states they are fulfilled. In the corresponding states a fresh proposition is labelled and the subformula of ϕ is replaced by this proposition. The PSPACE-hardness carries over from LTL model checking as well.

Fragments of this problem overlap on the one hand with parts of the LTL model checking problem (and its investigated restrictions in Section 6) and on the other hand with versions of the propositional model checking problem (i.e., Boolean formula evaluation). For the first part the relevant fragments are all those containing only operator subsets of $\{\mathsf{X},\mathsf{F},\mathsf{G},\mathsf{U},\mathsf{R}\}$ and for the latter subsets of $\{\mathsf{A},\mathsf{E}\}$. Hence these types of fragments are not considered any more. Further a complete classification of every relevant fragment is not available yet. In this survey we will present a Θ_2^{P}-hardness result for the monotone fragment with operators $\{\mathsf{E},\mathsf{A},\mathsf{X}\}$. Furthermore an initial step of the classification is done in [21]. There, several known implications from LTL results are transferred to the CTL* case.

Theorem 7.1 CTL*-MC$(\mathsf{E},\mathsf{A},\mathsf{X},\wedge,\vee)$ *is* Θ_2^{P}*-hard.*

Proof. We give a reduction from ODDS. Let $(\varphi_1,\ldots,\varphi_\ell)$ be a tuple of formulas in 3CNF. The construction of of a Kripke model from a 3CNF formula used in the NP-hardness for LTL-MC$(\mathsf{X},\wedge,\vee)$ (see Figure 11) can be extended here. Let $x_1,\ldots,x_n$ be the atoms used in all formulas, and w.l.o.g. each formula has m clauses. Let K_{φ_i} be the Kripke model constructed for φ_i as in Figure 11, where w_{φ_i} is the initial state. Figure 14 shows, how these parts are put together. For our new Kripke model K, we take all the K_{φ_i} and add a new initial state s that has edges to all initial states $w^{\varphi_1}, w^{\varphi_2},\ldots,w^{\varphi_\ell}$. The assignment ξ of K is obtained from all assignments of the K_{φ_i} and additionally assigns the new atom w_i to $\{w^{\varphi_i}\}$ for all i. Every path π starting in w_{φ_i} with $K,\pi \models \bigwedge_{i=1}^{m}\bigvee_{j=1}^{n}\mathsf{X}^j c_i$ corresponds to a satisfying assignment for φ_i. Accordingly, every path π starting in w_{φ_i} with $K,\pi \models \neg\bigwedge_{i=1}^{m}\bigvee_{j=1}^{n}\mathsf{X}^j c_i$ corresponds to a non-satisfying assignment for φ_i. In order to be able to express the latter without using $\neg$, we extend the assignment function ξ, that assigns atom c_i to states that correspond to an assignment that satisfies the ith clause in the formula, to additional atoms $\bar{c}_i$, that are assigned to states that do not satisfy the ith clause in the formula. Notice that every state satisfies either c_i

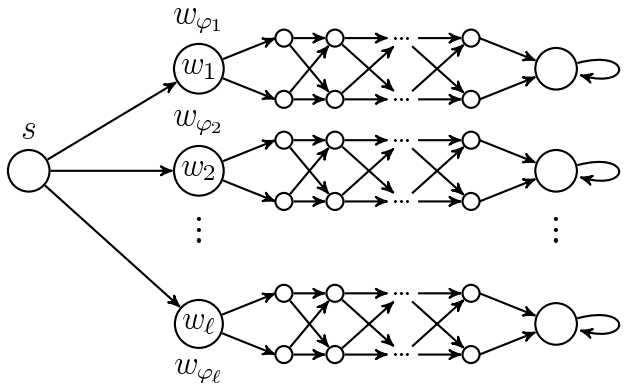


Figure 14: Excerpt of the Kripke model K for Theorem 7.1.

or $\bar{c}_i$. Then every path π starting in w_{φ_i} with $K, \pi \models \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} \mathsf{X}^j \bar{c}_i$ corresponds to a non-satisfying assignment for φ_i.

In order to state that $(\varphi_1, \ldots, \varphi_\ell)$ is in ODDS, we must say that for some odd $i < \ell$ holds $\varphi_1, \varphi_2, \ldots, \varphi_i \in$ SAT and $\varphi_{i+1} \notin$ SAT. This is done as follows.

$$\psi = \bigvee_{i \text{ odd}, i<\ell} \Big[\Big(\bigwedge_{r=1}^{i} \mathsf{E}\,\mathsf{X}(w_r \wedge \bigwedge_{q=1}^{m} \bigvee_{j=1}^{n} \mathsf{X}^j c_q) \Big) \wedge \Big(\mathsf{E}\,\mathsf{X}(w_{i+1} \wedge \mathsf{A} \bigvee_{q=1}^{m} \bigwedge_{j=1}^{n} \mathsf{X}^j \bar{c}_q) \Big) \Big]$$

It now holds that $(\varphi_1, \ldots, \varphi_\ell) \in$ ODDS if and only if $K, s \models \psi$. □

Now, in contrast to CTL, we have seen how powerful the logic CTL* is. The syntactical separation of path quantifiers E and A from the temporal operators X, F, G, R, U makes the model checking problem computationally harder (unless some collapses within the polynomial hierarchy happen).

For the next section we will finally leave the area of temporal logics and will visit one very young logic that is able to talk about dependencies of atomic propositions.

8 Modal Dependence Logic

The language of modal dependence logic (MDL) extends the language of modal logic by the dependence atom $\mathsf{dep}(p_1, \ldots, p_n)$, which intuitively states that the value of the atom p_n depends only on the values of $p_1, \ldots, p_{n-1}$, i.e., p_n is functionally determined by $p_1, \ldots, p_{n-1}$. Modal dependence logic was first introduced 2007 by Väänänen [38]. Unlike in usual modal logic, an MDL

formula is typically evaluated not in a single state but in a set of states (in this context called *team*), and this is different from evaluating the formula in each state separately. Therefore we generalize the definition of modal logic semantics as follows. Let $\mathcal{M} = (W, R, \xi)$ be a Kripke model and $T \subseteq W$ a team.

$$
\begin{array}{lll}
\mathcal{M}, T \models p & \text{iff} & \forall s \in T : s \in \xi(p),\ p \in \mathrm{PROP}, \\
\mathcal{M}, T \models \overline{p} & \text{iff} & \forall s \in T : s \notin \xi(p),\ p \in \mathrm{PROP}, \\
\mathcal{M}, T \models \mathsf{dep}(\vec{p}, q) & \text{iff} & \forall s_1, s_2 \in T : \\
 & & \text{if } \xi^{-1}(s_1) \cap \{\vec{p}\} = \xi^{-1}(s_2) \cap \{\vec{p}\} \\
 & & \text{then } s_1 \in \xi(q) \Leftrightarrow s_2 \in \xi(q), \\
\mathcal{M}, T \models \overline{\mathsf{dep}(\vec{p}, q)} & \text{iff} & T = \emptyset, \\
\mathcal{M}, T \models \neg\varphi & \text{iff} & \mathcal{M}, T \not\models \varphi, \\
\mathcal{M}, T \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{M}, T \models \varphi_1 \text{ and } \mathcal{M}, T \models \varphi_2, \\
\mathcal{M}, T \models \varphi_1 \vee \varphi_2 & \text{iff} & \exists T_1 \cup T_2 = T : \mathcal{M}, T_1 \models \varphi_1 \text{ and } \mathcal{M}, T_2 \models \varphi_2.
\end{array}
$$

The semantics of the operator $\vee$ (in dependence logic terms called *split-junction*) may look strange at first sight. However, when restricted to singleton teams T we coincide with the definition of usual disjunction. Note that the usual duality between $\wedge$ and $\vee$ is not given here. Instead we define the dual operator to $\wedge$ by introducing $\otimes$.

$$\varphi_1 \otimes \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2).$$

Then $\mathcal{M}, T \models \varphi_1 \otimes \varphi_2$ if and only if $\mathcal{M}, T \models \varphi_1$ or $\mathcal{M}, T \models \varphi_2$.

Before we define the MDL semantics for the modal operators we have to look at what a successor of a team is. On the way towards a definition of the modality $\Box$ the set of all successors of $T \subseteq W$ in $\mathcal{M} = (W, R, \xi)$ formally is written as

$$R(T) := \{s \in W \mid \exists s' \in T : (s', s) \in R\}.$$

For defining the modality $\Diamond$, we will consider subsets of the team of all successors of the given team, hence this is:

$$R\langle T \rangle := \{T' \subseteq R(T) \mid \forall s \in T\ \exists s' \in T' : (s, s') \in R\}.$$

Now we are ready to state the MDL semantics for the modal operators as follows.

$$
\begin{array}{lll}
\mathcal{M}, T \models \Diamond\varphi & \text{iff} & \exists T' \in R\langle T \rangle : \mathcal{M}, T' \models \varphi \\
\mathcal{M}, T \models \Box\varphi & \text{iff} & \mathcal{M}, R(T) \models \varphi
\end{array}
$$

Similar to the above, the usual duality between $\Diamond$ and $\Box$ does not hold in MDL. We define a dual operator to $\Diamond$ as follows.

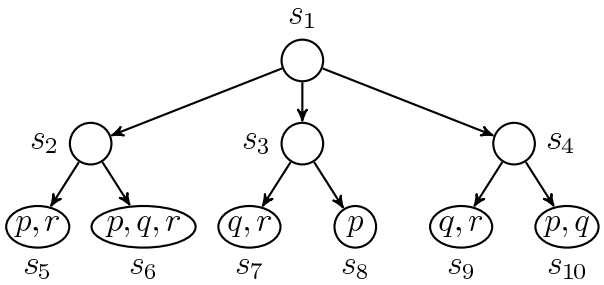
$$\boxdot\varphi := \neg\Diamond\neg\varphi.$$

The definition of the semantics of MDL is a direct generalization (more formally, a conservative extension) of the semantics of modal logic. For formula φ over the operator fragment $\{\Diamond, \Box, \wedge, \vee, \dot{-}\}$, we have

$$\mathcal{M}, T \models \varphi \Leftrightarrow \forall s \in T : \mathcal{M}, s \models \varphi.$$

In dependence logic, this property of φ is called *flatness*. The power of dependence logic appears when additionally the dependence atom is used.

We illustrate the expressive power of modal dependence logic compared to modal logic. Consider the following Kripke model $\mathcal{M}$.



We want to evaluate the formula

$$\Box\Diamond\mathsf{dep}(p, q, r)$$

in $\mathcal{M}$ on the starting team $T = \{s_1\}$. By evaluating the $\Box$ operator we have to check $\Diamond\mathsf{dep}(p, q, r)$ on $\{s_2, s_3, s_4\}$, the team $R(\{s_1\})$ of all successor of $\{s_1\}$. This subformula is satisfied if and only if there exists a valid successor team of $\{s_2, s_3, s_4\}$ on which the proposition r is functionally determined by the propositions p and q, i.e., $r = f(p, q)$ for some binary Boolean function f. Such a function exists if we choose the successor team $\{s_5, s_8, s_9\}$. On this team the dependence holds by picking f to be the exclusive or, i.e., $r = p \oplus q$.

For the MDL operator fragment $\{\Box, \Diamond, \dot{-}, \vee, \wedge, \mathsf{dep}\}$ Sevenster [34] showed that MDL formulas are expressible in modal logic with exponentially larger formulas. He also showed that this exponential blowup is necessary. We illustrate this blowup for the example MDL formula $\Box\Diamond\mathsf{dep}(p, q, r)$ from above. As we have seen, the dependence atom is true if there exists a function which determines r by p and q. We can simulate this by a disjunction over all possible Boolean functions

$$\bigvee_{f\in\mathbb{B}^2} \Box\Diamond(f(p,q)\leftrightarrow r)\ .$$

Because the number of all possible Boolean functions with arity n is 2^{2^n}, this formula is exponentially larger. At last step we can transform this formula directly into the modal logic formula

$$\begin{aligned}
&\Box\Diamond(p\wedge q\leftrightarrow r)\vee\Box\Diamond(\overline{p}\wedge q\leftrightarrow r)\vee\Box\Diamond(p\wedge\overline{q}\leftrightarrow r)\vee\\
&\Box\Diamond(\overline{p}\wedge\overline{q}\leftrightarrow r)\vee\Box\Diamond(p\vee q\leftrightarrow r)\vee\Box\Diamond(\overline{p}\vee q\leftrightarrow r)\vee\\
&\Box\Diamond(p\vee\overline{q}\leftrightarrow r)\vee\Box\Diamond(\overline{p}\vee\overline{q}\leftrightarrow r)\vee\Box\Diamond(p=q\leftrightarrow r)\vee\\
&\Box\Diamond(p\oplus q\leftrightarrow r)\vee\Box\Diamond(p\leftrightarrow r)\vee\Box\Diamond(\overline{p}\leftrightarrow r)\vee\Box\Diamond(q\leftrightarrow r)\vee\\
&\Box\Diamond(\overline{q}\leftrightarrow r)\vee\Box\Diamond(\bot\leftrightarrow r)\vee\Box\Diamond(\top\leftrightarrow r).
\end{aligned}$$

The model checking problem for modal dependence logic is defined as follows.

Problem:	MDL-MC(O)
Description:	the model checking problem for MDL.
Input:	A modal formula φ using operators in $O\subseteq\{\Box$, $\Diamond$, $\boxdot$, $\circledcirc$, $\vee$, $\wedge$, $\dot{-}$, $\neg$, $\mathsf{dep}(\cdot)\}$, a Kripke model $\mathcal{M}=(W,R,\xi)$, and an initial team $T\subseteq W$.
Question:	Does $\mathcal{M},T\models\varphi$ hold?

The computational complexity of model checking for the MDL fragment $\{\Box,\Diamond,\boxdot,\circledcirc,\vee,\wedge,\dot{-},\neg,\mathsf{dep}(\cdot)\}$ and some subfragments were studied by Ebbing and Lohmann in [11] and by Müller in [22]. Figure 15 illustrates some of their results. Note that, in difference to the complexity classifications in the previous sections, here we do not have the full lattice of all clones that contain constants, but $\oplus$ is missing. This logical connective has not been studied in dependence logic so far, and in fact there are several different sensible ways to define its semantics. Therefore, Figure 15 is not understood in the lattice-sense as before, but just as a summary of complexity results for some important sets of operators. For the remaining fragments a complete classification is pending.

In the rest of this section, we briefly sketch the general upper bound and then turn to two hardness proofs.

Theorem 8.1 *[22]* MDL-MC($\Diamond,\boxdot,\Box,\wedge,\circledcirc,\vee,\dot{-},\neg,\mathsf{dep}(\cdot)$) *is in* PSPACE.

The model checking algorithm for MDL uses an alternating polynomial-time Turing machine that evaluates an MDL formula by using existential guesses to simulate $\Diamond$ and splitjunction, and universal guesses to simulate $\boxdot$.

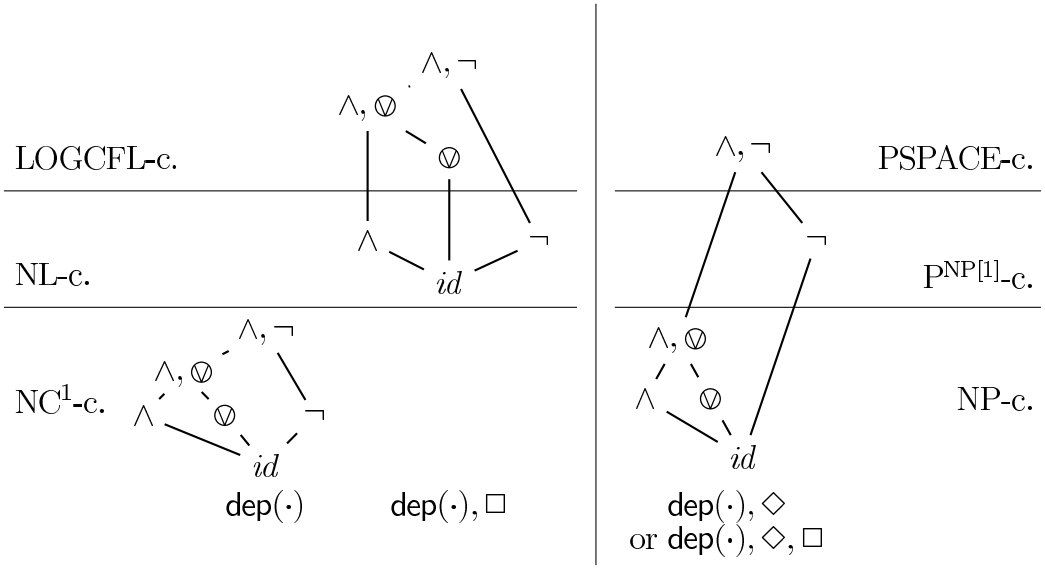


Figure 15: Complexity results for model checking modal dependence logics [11, 22]. On the left/right side the tractable/intractable fragments are shown.

Theorem 8.2 *[22]* MDL-MC$(\Diamond, \wedge, \neg)$ *is* PSPACE-*complete.*

Proof. The proof consists of a reduction from the value problem for quantified Boolean formulas QBF-VAL to the modal dependence logic model checking problem. Intuitively we will express the existential and universal quantifications of QBF-VAL formulas by the alternation of the modal operators "$\Diamond$" and "$\boxdot$". Teams are used to represent the propositional assignments, under which the formula part of the QBF-VAL instance has to be evaluated. As an interesting fact we will see that we do not need the dependence atom to obtain the hardness result.

For this purpose let $\varphi = \exists x_1 \forall x_2 \ldots \exists x_n \bigwedge_{i=1}^{m} (l_{i1} \vee l_{i2} \vee l_{i3})$ be a QBF-VAL instance, where all literals l_{ij} are over the quantified variables $x_1, \ldots, x_n$ and w.l.o.g. n is assumed to be odd. The reduction function maps φ to a MTL-MC instance $\langle \mathcal{M}, T_0, \delta_1 \rangle$, where $\mathcal{M} = (W, R, \xi)$. At first we give the construction of $\mathcal{M}$, see Figure 16 for an example. For each quantifier we will construct one connected component. In these connected components we will simulate the nesting of the quantified variable x_i by *delay states* d_i and the quantified value of the variable by *value states* x_i and $\neg x_i$:

$$W := \bigcup_{i=1}^{n} \left(\{ d_i^j \mid 1 \leq j \leq i \} \cup \{ x_i, \neg x_i \} \right).$$

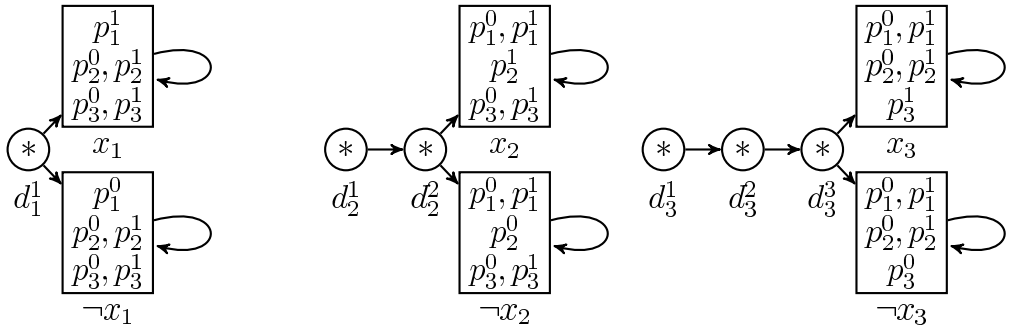


Figure 16: Kripke model for the QBF formula $\exists x_1 \forall x_2 \exists x_3 (x_1 \vee \neg x_2 \vee x_3)$.

For the quantified variable x_i the decision for the variables value will be made at point i. Before these decision points all delay nodes and all value nodes are connected in natural order. At the decision point i the delay state will branch to the two possible value states.

$$R := \bigcup_{i=1}^{n} \big(\{ (d_i^j, d_i^{j+1}) \mid 1 \leq j < i \} \cup \{ (d_i^i, x_i), (d_i^i, \neg x_i), (x_i, x_i), (\neg x_i, \neg x_i) \} \big).$$

At last we define the assignment for the atoms p_i^0 and p_i^1 (for $1 \leq i \leq n$), that stand for "x_i is assigned value 0 resp. 1". The idea is that in any team $T \subseteq W$, if $x_i \in T$ then $T \models p_i^1$ and $T \not\models p_i^0$, and accordingly if $\neg x_i \in T$ then $T \not\models p_i^1$ and $T \models p_i^0$.

$$\begin{aligned} \xi(p_i^1) &:= \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} - \{\neg x_i\} \\ \xi(p_i^0) &:= \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} - \{x_i\} \\ \xi(d_i^j) &:= \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} \end{aligned}$$

The starting team T_0 is the set of starting points of all components.

$$T_0 := \{d_i^1 \mid 1 \leq i \leq n\}.$$

We define the MDL formula inductively. Let us begin from the end, where a "final" team consists of non-contradictory value states, e.g. $\{x_1, \neg x_2, x_3\}$. Now the CNF has to be evaluated under the propositional assignment corresponding to such a team. By the properties of the assignment, $(p_a^1 \otimes p_b^0)$ is satisfied by a final team T iff $x_a \in T$ or $\neg x_b \in T$, what means that the assignment according to T satisfies $(x_a \vee \neg x_b)$. This shows how the clauses of the QBF-VAL instance have to be transformed.

Now for the quantifiers of the QBF-VAL instance. In our example, a formula $\Diamond \alpha$ is satisfied by the initial team $T_0 = \{d_1^1, d_2^1, d_3^1\}$ iff α is satisfied by

$\{x_1, d_2^2, d_3^2\}$ or by $\{\neg x_1, d_2^2, d_3^2\}$ or by $\{x_1, \neg x_1, d_2^2, d_3^2\}$. The latter team cannot be "continued" to a final team and should be excluded. It is the only of the three teams that does not satisfy $p_1^1 \otimes p_1^0$. Therefore, $\Diamond((p_1^1 \otimes p_1^0) \wedge \alpha)$ is satisfied by T_0 iff it is satisfied on one of the first two above teams. This construction can be used to simulate an existential quantifier in the QBF-VAL instance. The construction for the universal quantifier is accordingly. This yields the following inductive definition of the formulas δ_i.

$$\delta_i := \begin{cases} \Diamond((p_i^1 \otimes p_i^0) \wedge \delta_{i+1}) & \text{if } i \leq n \text{ odd}, \\ \boxdot(\neg(p_i^1 \otimes p_i^0) \otimes \delta_{i+1}) & \text{if } i \leq n \text{ even}, \\ \bigwedge_{i=1}^m (\hat{l}_{i1} \otimes \hat{l}_{i2} \otimes \hat{l}_{i3}) & \text{if } i = n+1. \end{cases}$$

where $\hat{l}_{ij} = p_\ell^1$ if $l_{ij} = x_\ell$, and $\hat{l}_{ij} = p_\ell^0$ if $l_{ij} = \neg x_\ell$.

Then we have that $\varphi \in$ QBF-VAL if and only of $\mathcal{M}, T_0 \models \delta_1$. □

For an example suppose that $\exists x_1 \forall x_2 \exists x_3 (x_1 \vee \overline{x}_2 \vee x_3)$ is a QBF-VAL instance. The corresponding Kripke model is that of Figure 16, on which the following must be checked.

$$\{d_1^1, d_2^1, d_3^1\} \models \Diamond((p_1^1 \otimes p_1^0) \wedge \ \boxdot(\neg(p_2^1 \otimes p_2^0) \otimes \ \Diamond((p_3^1 \otimes p_3^0) \wedge [p_1^1 \otimes p_2^0 \otimes p_3^1]))).$$

The role of negation $\neg$ in modal dependence logic is different from that in modal or temporal logics. This is a reason why negation and atomic negation are dealt as different operations. Whereas for the other logics considered in this survey, model checking for fragments with $\neg$ can be reduced to model checking for other fragments without $\neg$—e.g. for modal logic model checking for the $\{\Diamond, \neg\}$-fragment can be reduced to model checking for the $\{\Diamond, \Box\}$-fragment—this is not possible for modal dependence logic. Eventually, we consider a fragment of modal dependence logic with atomic negation only. The combination of $\Diamond$ and the dependence atom makes model checking intractable.

Theorem 8.3 *[11] Let* $\{\Diamond, \mathsf{dep}(\cdot)\} \subseteq O \subseteq \{\Diamond, \Box, \wedge, \otimes, \vee, \bar{\cdot}, \mathsf{dep}(\cdot)\}$ *be a set of modal dependence operators. Then* MDL-MC(O) *is* NP*-complete.*

Proof. For the hardness proof, we will reduce 3SAT to MDL-MC($\Diamond, \mathsf{dep}(\cdot)$). Consider an instance $\varphi = \bigwedge_{i=1}^m (l_{i1} \vee l_{i2} \vee l_{i3})$ of 3CNF with variables $x_1, \ldots, x_n$. It is transformed to an MDL model checking instance as follows. The Kripke model $\mathcal{M}_\varphi = (W, R, \xi)$ has one state for each clause of φ, and one state for each literal over $x_1, \ldots, x_n$:

$$W := \{C_i \mid 1 \leq i \leq m\} \cup \{x_i, \neg x_i \mid 1 \leq i \leq n\}.$$

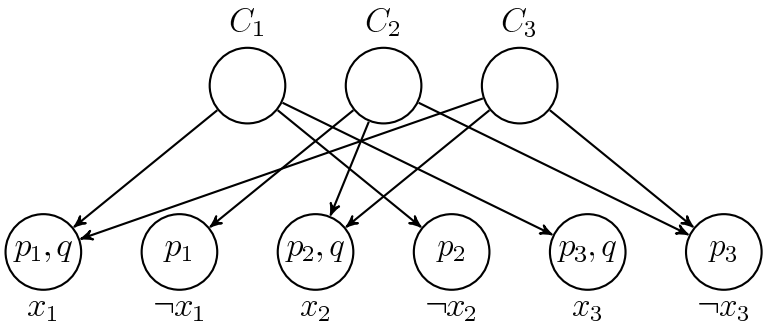


Figure 17: Kripke model $\mathcal{M}_\varphi$ for the CNF formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$.

Edges go from each clause states C_i to literal state l_j, if literal l_j appears in clause C_i:

$$R := \{(C_i, l_j) \mid l_i \text{ appears in } C_j\}.$$

The assignment maps atom p_i to states x_i and $\neg x_i$, and atom q to each x_i:

$$\xi(p_i) := \{x_i, \neg x_i\} \text{ for } 1 \leq i \leq n,$$
$$\xi(q) := \{x_i \mid 1 \leq i \leq n\}.$$

Figure 17 shows an example of the constructed model $\mathcal{M}_\varphi$. Notice that, e.g., on team $T = \{x_1, \neg x_1\}$ the dependence atom $\mathsf{dep}(p_1, q)$ is not satisfied, since p_1 is satisfied in both states of T, but q is satisfied only in one state. This can be generalized to the observation, that a team $T \subseteq \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ satisfies $\mathsf{dep}(p_1, \ldots, p_n, q)$ only if the team does not contain two states that stand for contrary literals x_i and $\neg x_i$. Such a $\mathsf{dep}(p_1, \ldots, p_n, q)$ satisfying team clearly defines an assignment to the atoms $x_1, \neg x_1, \ldots, x_n, \neg x_n$. If such a satisfying team is a successor set of the clause states $\{C_1, \ldots, C_m\}$, then every clause is satisfied by the corresponding assignment. Therefore, φ is satisfiable if and only if $\mathcal{M}, \{C_1, \ldots, C_m\} \models \Diamond \mathsf{dep}(p_1, \ldots, p_n, q)$.

The upper bound NP follows from the fact that existential quantifiers in the semantics of $\Diamond$ and $\vee$ can straightforwardly be "implemented" by guessing the right teams. □

References

[1] Michael Bauland, Martin Mundhenk, Thomas Schneider, Henning Schnoor, Ilka Schnoor, and Heribert Vollmer. The tractability of model checking for LTL: The good, the bad, and the ugly fragments. *ACM Trans. Comput. Log.*, 12(2):26, 2011.

[2] Olaf Beyersdorff, Arne Meier, Martin Mundhenk, Thomas Schneider, Michael Thomas, and Heribert Vollmer. Model checking CTL is almost always inherently sequential. *Logical Methods in Computer Science*, 7(2), 2011.

[3] Elmar Böhler, Nadia Creignou, Steffen Reith, and Heribert Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *SIGACT News*, 34(4):38–52, 2003.

[4] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings 19th Symposium on Theory of Computing*, pages 123–131. ACM Press, 1987.

[5] Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*. Clarendon Press, Oxford, 1997.

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

[7] Edmund M. Clarke. The birth of model checking. In Grumberg and Veith [14], pages 1–26.

[8] Edmund M. Clarke and E. Allen Emerson. Desing and synthesis of synchronisation skeletons using branching time temporal logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Verlag, 1981.

[9] Gerard R. Renardel de Lavalette, Alex Hendriks, and Dick de Jongh. Intuitionistic implication without disjunction. *J. Log. Comput.*, 22(3):375–404, 2012.

[10] Stéphane Demri and Phillipe Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.

[11] Johannes Ebbing and Peter Lohmann. Complexity of model checking for modal dependence logic. Technical report, CoRR, abs/1104.1034v1, 2011. To appear in Proc. Ninth International Tbilisi Symposium on Language, Logic and Computation.

[12] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18(2):194–211, 1979.

[13] Massimo Franceschet and Maarten de Rijke. Model checking hybrid logics (with an application to semistructured data). *J. Applied Logic*, 4(3):279–304, 2006.

[14] Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking – History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[15] Andreas Krebs. Model checking for LTL fragments with $\oplus$ is NP-hard, 2012. Personal communication.

[16] Saul A. Kripke. Semantical analysis of modal logic I. Normal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[17] Francois Laroussinie. About the expressive power of CTL combinators. *Information Processing Letters*, 54(6):343–345, 1995.

[18] Clarence I. Lewis. *A Survey of Symbolic Logic*. University of California Press, Berkley, 1918.

[19] Harry R. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.

[20] Nicolas Markey. Past is for free: on the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6-7):431–458, 2004.

[21] Arne Meier. *On the Complexity of Modal Logic Variants and their Fragments*. PhD thesis, Leibniz Universität Hannover, Institut für Theoretische Informatik, November 2011.

[22] Julian-Steffen Müller. Complexity of model checking for modal team logic. Technical report, CoRR, 2012.

[23] Martin Mundhenk and Felix Weiß. The complexity of model checking for intuitionistic logics and their modal companions. In *Proc. 4th Int. Workshop on Reachability Problems*, volume 6227 of *LNCS*, pages 146–160. Springer, 2010.

[24] Martin Mundhenk and Felix Weiß. The model checking problem for intuitionistic propositional logic with one variable is AC^1-complete. In *Proc. 28th STACS*, volume 9 of *LIPIcs*, pages 368–379. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[25] Martin Mundhenk and Felix Weiß. Intuitionistic implication makes model checking hard. *Logical Methods in Computer Science*, 8(2), 2012.

[26] Iwao Nishimura. On formulas of one variable in intuitionistic propositional calculus. *J. of Symbolic Logic*, 25:327–331, 1960.

[27] Amir Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE Computer Society Press, 1977.

[28] Emil Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

[29] Arthur N. Prior. *Time and Modality*. Clarendon Press, Oxford, 1957.

[30] Arthur N. Prior. *Past, Present, and Future*. Clarendon Press, Oxford, 1967.

[31] Thomas Schneider. *The Complexity of Hybrid Logics over Restricted Classes of Frames*. PhD thesis, Univ. of Jena, 2007.

[32] Philippe Schnoebelen. The complexity of temporal logic model checking. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic*, pages 393–436. King's College Publications, 2002.

[33] Henning Schnoor. The Complexity of Model Checking for Boolean Formulas. *International Journal of Foundations of Computer Science*, 21(03):289, 2010.

[34] Merlijn Sevenster. Model-theoretic and computational properties of modal dependence logic. *Journal of Logic and Computation*, 19(6):1157–1173, 2009.

[35] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[36] Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979.

[37] Michael Thomas. On the applicability of Post's lattice. *Inf. Process. Lett.*, 112(10):386–391, 2012.

[38] Jouko Väänänen. Modal dependence logic. In Krzysztof R. Apt and Robert van Rooij, editors, *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 237–254. Amsterdam University Press, 2008.

[39] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. LICS*, pages 332–344. IEEE Computer Society, 1986.

[40] Albert Visser. A propositional logic with explicit fixed points. *Studia Logica*, 40:155–175, 1980.

[41] Klaus W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19:833–846, 1990.