Master's Thesis

Complexity of Temporal Logics

Arne Meier

2007-11-09

Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Danksagung

Zunächst möchte ich mich sehr herzlich bei meinen Eltern bedanken, die mir während des gesamten Studiums in allen Bereichen zur Seite standen, mich unterstützten und in meinem Tun bekräftigten. Ohne Euch und Euren Rat hätte ich nicht Informatik studieren können! Vielen Dank.

Besonderer Dank geht auch an meine Freundin Julia, die mir in allen Belangen liebevoll geholfen hat und in jeder Prüfung alle Daumen drückte. Vielen Dank mein Engel.

Desweiteren möchte ich mich noch bei meinem zukünftigen Doktorvater Heribert Vollmer für die Betreuung bei dieser Abschlussarbeit bedanken.

Abschließend geht mein Dank an meine Mitstudenten und Weggefährten während des Studiums. Vielen Dank für Eure Erklärungen und das gemeinsame Lernen während des Semesters und während der Prüfungszeit. Insbesondere möchte ich hierbei Sebastian, Dennis, Daniel, Thilo und Moritz danken. Ich hoffe wir bleiben weiterhin im Kontakt!

Abstract. In 2000, Emerson and Jutla finally showed 2-EXPTIME-completeness for the Satisfiability problem for the temporal logic CTL*. We investigate fragments of this logic, that are restricted to the single temporal operator ne**X**t and both path operators. With a straight reduction from QBF-3SAT we are able to state PSPACE-hardness for this restricted logic. A similar proof for CTL and a PSPACE-algorithm leads to PSPACE-completeness for CTL restricted in the same way. With the help of Post's Lattice, we determine the computational complexity of these logics for a huge part of the lattice. On the one hand we discover PSPACE-hard (resp. PSPACE-complete) results, and on the other hand results for L.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

In this introduction we will describe the field of temporal logics and the structure of Post's lattice in a more elaborative way, as it is essential for this master's thesis and it is not a well known area in computational complexity theory.

In 1977, Pnueli introduced linear temporal logic (LTL) in [Pnu77] for the specification and verification of reactive systems (e.g., a computer system, system properties or a hardware architecture) which yields importance in computer science. As temporal logics are quite more complicated than propositional logic, we will motivate the benefit that is gained by them. Because of the results in the 1930s in recursion theory by Goedel, Church, Kleene, Turing and Post (for more informations concerning these theories, confer [Dav04] by Martin Davis), we know that programs like a specification-checker, an endless-loop-detector or a code-optimizer cannot exist. This is a cruel fact since otherwise we could write programs that remove all bugs in our written code. Thus the motivation is to analyze this irrevocable situation and make restrictions to the given system (i.e., algorithm) for getting some partial results.

As mentioned in [CGP99], the first step in verifying the correctness of a system is specifying the properties that the system should have. The common used model to describe such a system is the *Kripke structure*, that is a form of a finite state machine. It consists of four different parts. First we have a finite set of states S that describe the different conditions of a *reactive system*. Reactive Systems normally do not terminate, in fact, they interact very frequently with their environment through many inputs. Thus we cannot determine a function-like input-output behavior (else we could use verification methodologies like the Hoare logic or the inductive method of Floyd; see additional informations in [GMH81] and [Hoa69]): for such a system we investigate the behavior over time. For that we must be able to specify the transitions between the states. This is the second component of a Kripke structure: a *total*, binary relation R over the set of given states. Thus, for every state $s \in S$ we have at least one defined successor state $s' \in S$ with $R(s, s')$. Now we can describe a kind of computation of a given reactive system in terms of its transitions. Each single state describes a snapshot of a reactive system. If we consider such a specific moment, we can investigate some kind of properties (where the set of all used atomic propositions is combined in the set $\mathcal{AP}$) that hold. The fourth and last component is a function P that labels each state with a set of properties which are true in this state. Now we can model reactive systems in a Kripke structure $M = (S, R, \mathcal{AP}, P)$.

If we want to verify the correctness of such a Kripke structure we need a formalism to describe a given specification. Hence we will take a more closer look at the temporal logics. As well known from propositional first order logic we have the Boolean functions *and*, *or* and *not* (resp., $\wedge, \vee, \neg$). We use these Boolean connectors to express formulas consisting of atomic propositions in a specific state. At the moment, we cannot describe anything beyond a single state, but we want to investigate the behavior of time in a

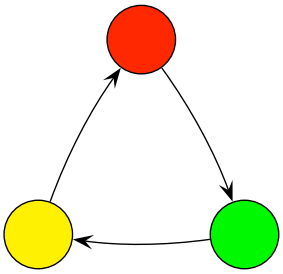


Figure 1.1: Kripke structure for a traffic light

given system (as mentioned above). Therefor, we must be able to specify that some property holds in all reachable states or some error value never holds. To describe such properties through a temporal logic formula, we need two temporal operators ($\mathbf{X}, \mathbf{U}$) and one path quantifier ($\mathbf{E}$). The remaining two temporal operators ($\mathbf{G}, \mathbf{F}$) and one path quantifier ($\mathbf{A}$) can be expressed through combinations of the former. At first, we need to remark that the properties are usually specified over an *infinite* path of transitions in the branching Kripke structure. Next, we will give some native linguistic meanings of these operators and quantifiers. If we want a property to hold in the second state of a given path, we use the ne**X**t operator. *Eventually* holding properties are described by the **F**uture operator. Now we can model **G**lobal properties that always hold, and a path on which in every preceding state the first property holds **U**ntil the second holds. Finally we can state wether there **E**xists a path that fulfills the given formula or on **A**ll possible paths the formula must hold.

Finally, we are able to define LTL-formulas. In short, we can combine the upper operators only in a way like $\mathbf{A}\varphi$, where φ is only composed of temporal and Boolean operators, but no **A**s or **E**s. To familiarize the reader with Kripke structures and temporal logic formulas we want to give a very easy example in Figure 1.1. Formally the definition of this Kripke structure is

$$M = (\{s_1, s_2, s_3\}, \{(s_1, s_2), (s_2, s_3), (s_3, s_1)\}, \{\blacksquare, \blacksquare, \blacksquare\}, P\}),$$

where $P(s_1) = \blacksquare$, $P(s_2) = \blacksquare$ and $P(s_3) = \blacksquare$. M is a very simple model for a typical traffic light.

Concerning traffic lights, the red and the green light should not be active in the same moment. A LTL-formula which describes that fact would be

$$\varphi := \mathbf{AG}\neg(\blacksquare \wedge \blacksquare),$$

and is read "On all paths the property $\blacksquare$ and $\blacksquare$ do not hold globally the same time" (which means in every state on each path).

Thus φ is fulfilled by the given Kripke structure M in the states $s_1, \ldots, s_3$, as in no state on the infinite path $\pi = s_1, s_2, s_3, s_1, \ldots$ the propositions $\blacksquare$ and $\blacksquare$ hold simultaneous – we write $M, s_1 \models \varphi$. Without using the **A**-quantifier, we can say: "on all possible paths π starting at s_1 in the Kripke structure M, it holds $\mathbf{G}\neg(\blacksquare \wedge \blacksquare)$".

The question we answered right now is the well-known Model-Checking problem you see in figure 1.2

Input:	Kripke structure $M = (S, R, \mathcal{AP}, P)$, temporal logic formula φ
Question:	is there a state $s \in S$ such that $M, s \models \varphi$?

Figure 1.2: Model-Checking problem for temporal logic

Input:	temporal logic formula φ
Question:	is there a Kripke structure $M = (S, R, \mathcal{AP}, P)$ and a state $s \in S$ such that $M, s \models \varphi$?

Figure 1.3: Satisfiability problem for temporal logic

Model-Checking is important for software development. Gerard Holzmann reveived the ACM Software System Award in the year 2001 for his model-checking system SPIN as "it has made advanced theoretical verification methods applicable to large and highly complex software systems" (for further informations read [Hol91] and [HP96] or consider the Website of the ACM[1]).

Unfortunately, the LTL-Model-Checking problem is PSPACE-complete as proven by Sistla and Clark in 1985 (see [SC85]). That is quite harder than we know from the problem of Satisfiability of propositional formulas SAT which is NP-complete (see [Coo71]). Thus we do not have an algorithm for LTL-Model-Checking that runs in polynomial time, if PSPACE $\neq$ P. Adjusting SAT to the temporal world, we get the Satisfiability problem for temporal logics you can see in Figure 1.3.

Fortunately, LTL-Satisfiability is not harder than LTL-Model-Checking – it stays PSPACE-complete. This is not the common consequence as we will see for the next temporal logic.

Computation Tree Logic (CTL) was first used by Clark and Emerson in [CE81] (and in a slightly different form by Quielle and Sifakis in [QS82]). The main difference to LTL is, that we can now nest the path quantifiers in any arbitrary way. But there is another restriction: before every temporal operator we need a single path quantifier (e.g., $\mathbf{AX}f$ or $\mathbf{EGAX}g$). The Model-Checking problem for CTL is very easy: it can be done in time linear in the size of the model and the length of the formula – and it is complete for the class P (see [CES86]). For CTL, the gap between the Model-Checking and Satisfiability problem is very huge. If we concern the Satisfiability problem for CTL-formulas, we become EXPTIME-complete which has been proven, in 1979/1980 independently, by Fischer and Ladner in [FL79] and by Pratt in [Pra80].

Finally, we make a last step over to the most powerful logic where none restrictions occur. Computation Tree Logic* combines the temporal logics LTL and CTL in one model and therefore is called informally the *full branching time logic*.

One year after Sistla and Clarke showed the PSPACE-completeness for LTL-Model-Checking, they expanded their result to CTL*-Model-Checking in [CES86]. Now a line of reductions exists, starting at the Satisfiability problem for *quantified Boolean formulas* (QBF) to LTL-Model-Shecking and LTL-Satisfiability over to CTL*-Model-Checking. The main question was, if the Satisfiability problem is PSPACE-complete, too. But four years later in 1990, Emerson showed in [Eme90] that CTL-Satisfiability is already

[1] `http://awards.acm.org/software_system`

temporal logic	Satisfiability	Model-Checking
CTL	EXPTIME-complete	P-complete
LTL	PSPACE-complete	PSPACE-complete
CTL*	2-EXPTIME-complete	PSPACE-complete

Table 1.1: Complexity of the problems Satisfiability and Model-Checking for temporal logics.

EXPTIME-complete. Emersons argumentation based on the concept of a bit different Turing Machine (Alternating Turing Machines). CTL*-Satisfiability was shown to be 2-EXPTIME-hard by Vardi and Stockmeyer in [VS85]. The containment in 2-EXPTIME was shown by Emerson and Jutla in [EJ00], which yields its 2-EXPTIME-completeness. As a short overview, the complexity situation of temporal logics is shown in Table 1.1.

The main intent of this thesis is to investigate fragments of the computation tree logics. We will make constraints, like disallowing specific Boolean functions, and analyze the complexity of the resulting logics. Our aim is to achieve a classification that is as complete as possible. Dealing with restrictions will help to get a better understanding for hardness and completeness results in complexity theory, and it will provide a more complete classification of the temporal logics than currently available. Analyzing fragments of a logic helps getting a more detailed point of view concerning its complexity-structure.

In 1979, Lewis was the first who applied such restrictions in a systematic way in [Lew79]. He established a dichotomy in Satisfiability for propositional logic (NP-completeness versus solvability in P). Another example is the discovered trichotomy in modal Satisfiability by Bauland et al in [BHSS06]. Later we will see that these results have a great impact on our studies about temporal logic fragments.

When talking about restrictions of Boolean functions the concept of clones must be introduced. A *clone* is a set of Boolean functions that is closed under *superposition* (i.e., it contains all projection functions and is closed under permutation, identification of variables, and arbitrary composition). One such clone is, e.g.,

$$E := \{f \mid f \text{ is a } n\text{-ary } \textit{and}\text{-function or a constant function}\},$$

whose base is $\{\wedge, 0, 1\}$. As E is closed under superposition, we can combine the logical *and* in any arbitrary way. Thus we get a n-ary function.

The complete classification regarding the restrictions of Boolean operators follows in the structure of Post's Lattice [Pos41]. Within this hierarchy Post showed which clones arise from combinations of other clones. When analyzing the complexity of logics restricted to a specific clone, you can use this structure to achieve upper and lower bounds of complexity. For more informations and definitions regarding Post's Lattice take a look at Chapter 2 and at Figure 2.4 on page 19.

In Chapter 3 we will only investigate the Satisfiability problem as its 2-EXPTIME (resp. EXPTIME) complexity is much more intractable than the inclusion in PSPACE for the Model-Checking problem (resp. containment in P for CTL). We will restrict the temporal logics CTL, CTL* and CTL$^+$ (which will later be defined) to the temporal

Input:	quantified Boolean formula φ in 3CNF
Question:	$\varphi \equiv 1$?

Figure 1.4: Quantified 3SAT problem

operator **X**. This restrictions results from a straight reduction starting at the Satisfiability problem of *quantified Boolean formulas*, respectively the version restricted to 3CNF (*conjunctive normal form* with three literals per clause), which can be seen in Figure 1.4. Hereby we show PSPACE-hardness for the three restricted CTL-versions which is one of our main results in this thesis. Additionally, we are able to state this hardness result for even formulas with only one single atomic proposition.

Concerning the restricted computation tree logic, we show containment in PSPACE which yields a completeness result for PSPACE. For these hardness results, we achieve a lower bound at the clone S_1. Referring to formulas restricted to D and R_1, we show that everything below these clones can easily decided in L, as all such formulas are trivially satisfiable and we only need to check syntactical correctness of a given formula. Finally, we achieve containment in L for CTL restricted to the clone N.

In Table 1.2 we show a summarized view of our achieved results in this thesis. Regarding the abbreviatory notation, "$>$" means "above" and "$<$" means "below" in the structure of Post's Lattice. The fields denoted with "?" are questions that are left open. All other not mentioned cases are also left open.

	BF or $> S_1$	$< R_1$ or $< D$	$< N$
CTL*-SAT	2-EXPTIME-complete	in L	?
CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)	PSPACE-hard	in L	?
CTL-SAT	EXPTIME-complete	in L	in L
CTL-SAT($\{\mathbf{AX}, \mathbf{EX}\}$)	PSPACE-complete	in L	in L
CTL$^+$-SAT	2-EXPTIME-complete	in L	?
CTL$^+$-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)	PSPACE-hard	in L	?

Table 1.2: Complexity results Overview

2 Preliminaries

In this chapter we will define all necessary mathematical notations, complexity classes, propositional and temporal logics, and we will give a brief introduction into the theory of clones and techniques with respect to Post's Lattice.

2.1 Notation

Let $\mathbb{N}$ denote the set of *natural numbers*, i.e., $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{N}^+ = \mathbb{N}\backslash\{0\}$. We use ε as the empty word. When talking about a *formal language* L, we define L over the *alphabet* $\Sigma = \{0, 1\}$. Any other non-single alphabet can be used through the usual reasons. Thus $\Sigma^\star$ is equal to $\mathbb{N}$, as we can sort $\Sigma^\star$ in length-lexicographical order and get natural numbers in binary representation (where the first element is ε).

Let $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ be a set of $n \in \mathbb{N}$ elements. Then we call $2^\mathcal{S}$ the *power set* of $\mathcal{S}$, i.e., $2^\mathcal{S} = \{\emptyset, \{s_1\}, \{s_2\}, \dots, \{s_n\}, \{s_1, s_2\}, \{s_1, s_3\}, \dots, \{s_1, s_n\}, \dots, \{s_1, \dots, s_n\}\}$, or shorter $2^\mathcal{S} = \{x \mid x \subseteq \mathcal{S}\}$. If A is a set over an alphabet Σ, then $A^C := \Sigma^\star\backslash A$ is the complement of A (instead A^C we write $\overline{A}$, too). With $|\mathcal{S}|$ we denote the cardinality of $\mathcal{S}$. Analogously we denote with $|s|$ the length of a given string $s = s_1 \dots s_n$.

Through this thesis we use a couple of standard mathematical abbreviations. Whenever we want to express the opposite of $=$, we write $\neq$ and in the same manner for $\in$, we write $\notin$. That is, we use "/" to denote the denial. With ■ we denote the end of a proof, with ■ the end of a proof idea, and with □ the end of a definition, observation, proposition, lemma or example. Mostly we write "iff" or sometimes "$\Leftrightarrow$" instead of "if and only if", or use "$\dots \Rightarrow \dots$" to abbreviate a sentence "If $\dots$, then $\dots$".

2.2 Formal Logic

In the next three subsections we will introduce the three logical concepts that are essential for this thesis. At first, we will give an introduction to the propositional logic. Then, we will define the concepts of temporal logics, and finally, we will define some basics in modal logic, that will help to understand similarities within proofs in both fields of temporal and modal logic, as many of our proofs will have their roots in the modal world.

2.2.1 Propositional Logic

We assume the reader is familiar with the well-known Boolean functions *and* ($\wedge$), *or* ($\vee$) and *implication* ($\rightarrow$), which we call *sentential connectives*, too. A *propositional variable* is defined over its two possible *logical values true* or *false* (resp., 1 or 0). In this thesis, we usually denote a propositional variable with the characters x, y, z or q. A *literal* is a propositional variable x or its *negation*, which we denote with $\neg x$ or

$\overline{x}$. A *formula* is a combination of literals through sentential connectives. Usually, we assume without loss of generality a formula to contain the variables $x_1, x_2, \ldots, x_n$ for $n \in \mathbb{N}$ (else we rename the variables to achieve such a form). Therefor we denote with $V_n := \{x_i \mid 1 \leq i \leq n, n \in \mathbb{N}\}$ the set of n variables, and denote with $\mathbb{F}_n$ the set of all propositional formulas with $n \in \mathbb{N}$ variables. If we do not need to refer to the number of variables, we omit writing $\mathbb{F}_n$ and use $\mathbb{F}$ instead. A *clause* C is a disjunction of literals, i.e., $C = l_1 \vee \cdots \vee l_n$ for $n \in \mathbb{N}$ and usually we assume $l_i \neq l_j$ for $1 \leq i, j \leq n$. We define a formula φ that is in *conjunctive normal form* (CNF) as a conjunction of clauses: $\varphi := C_1 \wedge \cdots \wedge C_m$ for $m \in \mathbb{N}$. In the same way we proceed for 3CNF and k-CNF, as a CNF-formula with $k \in \mathbb{N}$ literals per clause.

An assignment for a given formula φ is a total function $\theta \colon V_n \to \{0, 1\}$ that allocates logical values to variables. The set of all assignments for a formula φ with n variables is denoted with $\Theta_n := \{\theta \mid \theta \colon V_n \to \{0, 1\}\}$.

Now we define the semantic of the assignment function θ.

Definition 1 (Evaluation function) Let $n \in \mathbb{N}$, $\varphi \in \mathbb{F}_n$ and $\theta \in \Theta_n$ be an assignment. We define $\hat{\theta} \colon \mathbb{F}_n \to \{0, 1\}$ inductively by:

1. If $x \in V_n$, then $\hat{\theta}(x) = \theta(x)$.

2. Let $\varphi, \psi \in \mathbb{F}$ be formulas, then

$$\hat{\theta}(\varphi \wedge \psi) = \begin{cases} 1, & \text{if } \hat{\theta}(\varphi) = 1 \text{ and } \hat{\theta}(\psi) = 1 \\ 0, & \text{otherwise,} \end{cases}$$

$$\hat{\theta}(\varphi \vee \psi) = \begin{cases} 1, & \text{if } \hat{\theta}(\varphi) = 1 \text{ or } \hat{\theta}(\psi) = 1 \\ 0, & \text{otherwise,} \end{cases}$$

$$\hat{\theta}(\neg\varphi) = \begin{cases} 1, & \text{if } \hat{\theta}(\varphi) = 0 \\ 0, & \text{otherwise.} \end{cases}$$

Thus $\hat{\theta}(\varphi)$ denotes the logical value of φ. Usually, we omit writing $\hat{\theta}$ and use θ instead. □

Definition 2 (Model property) Let $n \in \mathbb{N}$, $\varphi \in \mathbb{F}_n$ and $\theta \in \Theta_n$. If $\hat{\theta}(\varphi) = 1$, we call θ a *model* of φ (or φ is *satisfied* by θ) and write $\theta \models \varphi$. If $\hat{\theta}(\varphi) = 0$, we write $\theta \not\models \varphi$. □

Finally we observe a property that associates a formula φ with another formula ψ concerning the model property.

Definition 3 (Logical Equivalence) Let $n \in \mathbb{N}$ and $\varphi, \psi \in \mathbb{F}_n$. If for each $\theta \in \Theta_n$ we have $\hat{\theta}(\varphi) = \hat{\theta}(\psi)$, we call φ *logical equivalent* to ψ and write $\varphi \equiv \psi$. □

Many proofs for our complexity results are based on reductions (we will define them later in Section 2.3), which are, according to this, a central part in our thesis. Therefor, we need to introduce a special form of propositional logic, that is quantified Boolean formulas. In Chapter 3, we will start our reductions beginning at the problem QBF-3SAT.

Definition 4 (Quantified Boolean Formulas) A *quantified Boolean formula* (qBf) is defined inductively as follows:

- 1 and 0 are qBfs.
- If x is a propositional variable, then x is a qBf.
- If φ_1, φ_2 are qBfs, then $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2$ are qBfs.
- If φ is a qBf and x is a propositional variable in φ, then $\exists x\varphi$ and $\forall x\varphi$ are qBfs.

The semantics of the first three cases is as in normal propositional logic. The last case's semantic is defined by

- $\exists x\varphi \equiv \varphi(0) \vee \varphi(1)$ and
- $\forall x\varphi \equiv \varphi(0) \wedge \varphi(1)$,

where $\varphi(i)$ is the formula φ, where all appearances of the variable x have been replaced by the logical value i, $i \in \{0, 1\}$. □

Without any loss of generality, we assume all given qBfs are in *prenex normal form*, where all quantifiers are at the beginning of the formula:

$$\exists x_1 \forall x_2 \ldots Q x_n F(x_1, \ldots, x_n),$$

and F is quantifier-free (i.e., neither $\exists$ nor $\forall$ appears in F).

Usually, we consider a qBf *fully quantified*, i.e., each variable is in the scope of some quantifier.

With these definitions, we can make an observation that shows the complexity of the quantified Boolean formulas.

Observation 1 *Every given qBf φ is equivalent to a propositional logic formula ψ, but its notation is a very abbreviated form of ψ (which is equal to either* true *or* false*). The length $|\psi|$ is exponential in size of the number of quantifiers in φ:*

$$\exists x_1 \forall x_2 \ldots Q x_n F(x_1, \ldots, x_n) \equiv \psi,$$

where ψ is a formula of length exponential in n, which can be easily shown by induction over n. □

If we consider the Satisfiability problem for qBf-formulas, then we investigate the following similarity to SAT:

$$\varphi \in \text{SAT } \textit{iff } \exists x_1 \ldots \exists x_n \varphi \equiv 1,$$

for $\varphi \in \mathbb{F}_n, n \in \mathbb{N}$. This is quite interesting, as the other side is equal to the set of formulas which are tautologies (i.e., given a formula $\varphi \in \mathbb{F}_n$ and for every assignment $\theta \in \Theta_n$, it holds $\theta \models \varphi$):

$$\varphi \in \text{TAUT } \textit{iff } \forall x_1 \ldots \forall x_n \varphi \equiv 1.$$

For a more elaborative introduction to propositional logic with many good examples and exercises we refer the reader to the textbook [End01].

2.2.2 Temporal Logic

In this subsection we will explain all preliminaries around temporal logics, as Kripke structures and the four examined logics in Chapter 3. For further informations and examples we refer the reader to the common textbook [CGP99], that had some influence on our definitions.

First we will start with the structure used to model the system that has to be analyzed.

Definition 5 (Kripke Structure) A *Kripke Structure* is a four-tupel $M = (S, R, \mathcal{AP}, P)$ where

- $S = \{s_1, s_2, \ldots, s_n\}$ is a finite set of *states* for $n \in \mathbb{N}$,
- $R \subseteq S \times S$ is a total *transition relation*, such that for every $s \in S$ there exists a $s' \in S$ with $(s, s') \in R$,
- $\mathcal{AP}$ is the set of *atomic propositions*, and
- $P \colon S \to 2^{\mathcal{AP}}$ is a *label function* that labels a set of atomic propositions to each single state. □

An example for such a Kripke structure is given in Figure 1.1 on page 2.

Observation 2 *A Kripke structure and a deterministic finite state machine have two similarities: a finite set of states and a total transition relation. By contrast, in Kripke structures we have additional properties that can hold in a specific state.* □

In temporal logics we construct formulas, that make statements about infinite paths in Kripke structures. Such paths are defined in the following definition.

Definition 6 Let $M = (S, R, \mathcal{AP}, P)$ be a Kripke structure. Then an infinite sequence of states π is a *path* (in M), i.e., $\pi := s_0, s_1, s_2, \ldots$ if for every $i \geq 0$, it holds $(s_i, s_{i+1}) \in R$.

We use π^i to denote the *suffix* of π starting at the i-th state s_i and $\pi(i)$ to denote the i-th state in π. □

Now we will turn towards the temporal operators that have been motivated in the introduction. At first, we will give some informal meanings of these operators. Afterwards in Definition 11, we will formally define them by its semantics.

- **X** means "next" and requires that a property holds in the second state of a given path.
- **F** means "future" or "eventually" and requires that a property holds in some state of a given path.
- **G** means "globally" and requires that a property holds in every single state of a given path.
- **U** means "until" and requires two properties, where $\varphi \mathbf{U} \psi$ means, that φ holds until a state in which ψ holds.

We need to remark, that in literature, sometimes the temporal operator **R** is defined as the dual operator to **U**. As we never use **R** in the appearing formulas in Chapter 3, we omit the definition.

Observation 3 *For the operator* **U** *it is possible, that the first property never holds.* □

Now after getting a look at the temporal operators, we will give native linguistic meanings for the two used *path quantifiers*:

- **A** means "all" and requires a property to hold in each state for every possible path starting at a given state.
- **E** means "exists" and requires a property to hold in each state for a possible path starting at a given state.

As we now understand how the temporal operators and path quantifiers "work", we can make an observation that summarizes the smallest class of operators, quantifiers and Boolean functions that can express everything.

Observation 4 *It suffices to use the operators* $\mathbf{X}, \mathbf{U}$ *and* $\mathbf{E}$*, and the sentential connectives* $\vee$ *and* $\neg$ *to express the remaining ones:*

- $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$
- $\mathbf{F}\varphi \equiv \mathit{true}\mathbf{U}\varphi$
- $\mathbf{G}\varphi \equiv \neg\mathbf{F}(\neg\varphi)$
- $\mathbf{A}\varphi \equiv \neg\mathbf{E}(\neg\varphi)$ □

Temporal logic formulas consist of two different types. On the one hand there are path formulas that hold along a specific path. On the other hand there are state formulas that hold in a specific state. Now we will define both types.

Definition 7 (State Formulas) Let $\mathcal{AP}$ be the set of atomic propositions. Then *state formulas* are usually defined inductively as follows:

- If $x \in \mathcal{AP}$, then x is a state formula.
- If φ and ψ are state formulas, then $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi$ and $\varphi \rightarrow \psi$ are state formulas.
- IF φ is a **path** formula, then $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are state formulas. □

Definition 8 (Path Formulas) Let $\mathcal{AP}$ be the set of atomic propositions. Then *path formulas* are usually defined inductively as follows:

- If φ is a state formula, then φ is a path formula.
- If φ and ψ are path formulas, then $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi, \mathbf{X}\varphi, \mathbf{F}\varphi, \mathbf{G}\varphi$ and $\varphi\mathbf{U}\psi$ are path formulas. □

Finally we are able to define the *full branching time logic*, that is CTL*.

Definition 9 (Computation Tree Logic CTL*) CTL* is the set of all state formulas, as defined above. □

Now there exist two relevant subsets of CTL*: on the one hand we have the *branching time logic* CTL, and on the other hand we have the *linear time logic* LTL. As the Linear Time Logic LTL is not investigated in our thesis, we omit the definition.

Definition 10 (Computation Tree Logic CTL) CTL is built from the eight operators

- **AX** and **EX**
- **AF** and **EF**,
- **AG** and **EG**,
- **AU** and **EU**.

Thus CTL-formulas consist of the Boolean sentential connectives and these eight temporal operators with preceding path quantifier. □

As above, we can make a similar observation concerning the smallest class of needed operators.

Observation 5 *Each of the eight operators can be expressed in terms of the three operators* **EX**, **EG** *and* **EU***:*

- $\mathbf{AX}\varphi = \neg\mathbf{EX}(\neg\varphi)$
- $\mathbf{EF}\varphi = \mathbf{E}(\mathit{true}\mathbf{U}\varphi)$
- $\mathbf{AG}\varphi = \neg\mathbf{EF}(\neg\varphi)$
- $\mathbf{AF}\varphi = \neg\mathbf{EG}(\neg\varphi)$
- $\mathbf{A}(\varphi\mathbf{U}\psi) = \neg\mathbf{E}\left(\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)\right) \wedge \neg\mathbf{EG}\neg\psi$ □

The two operators that are used most in this thesis are illustrated in Figure 2.1. At the moment, the models property is defined for the semantics of the propositional logic world (see Definition 2). Thus we need to extend this definition for the temporal world, which can be seen in the next definition.

Definition 11 (Temporal model property) Let φ_1, φ_2 be state formulas and ψ_1, ψ_2 be path formulas. Let $M = (S, R, \mathcal{AP}, P)$ be a Kripke structure, π be a path in M and $s \in S$ be a state. Let $x \in \mathcal{AP}$ be an atomic proposition. Then we define the relation $\models$ (for the temporal world) inductively as follows:

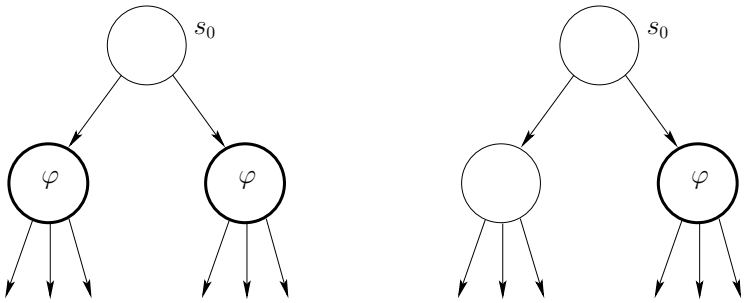


Figure 2.1: Most used CTL operators in Chapter 3

$M, s \models x$	$\Leftrightarrow$	$x \in P(s)$
$M, s \models \neg\varphi_1$	$\Leftrightarrow$	$M, s \not\models \varphi_1$
$M, s \models \varphi_1 \vee \varphi_2$	$\Leftrightarrow$	$M, s \models \varphi_1$ or $M, s \models \varphi_2$
$M, s \models \varphi_1 \wedge \varphi_2$	$\Leftrightarrow$	$M, s \models \varphi_1$ and $M, s \models \varphi_2$
$M, s \models \mathbf{E}\psi_1$	$\Leftrightarrow$	there exists a path π from s such that $M, \pi \models \psi_1$
$M, s \models \mathbf{A}\psi_1$	$\Leftrightarrow$	for every path π starting at s, it holds $M, \pi \models \psi_1$
$M, \pi \models \varphi_1$	$\Leftrightarrow$	s is the first state of π and $M, s \models \varphi_1$
$M, \pi \models \neg\psi_1$	$\Leftrightarrow$	$M, \pi \not\models \psi_1$
$M, \pi \models \psi_1 \vee \psi_2$	$\Leftrightarrow$	$M, \pi \models \psi_1$ or $M, \pi \models \psi_2$
$M, \pi \models \psi_1 \wedge \psi_2$	$\Leftrightarrow$	$M, \pi \models \psi_1$ and $M, \pi \models \psi_2$
$M, \pi \models \mathbf{X}\psi_1$	$\Leftrightarrow$	$M, \pi^1 \models \psi_1$
$M, \pi \models \mathbf{F}\psi_1$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models \psi_1$
$M, \pi \models \mathbf{G}\psi_1$	$\Leftrightarrow$	for all $i \geq 0$, it holds $M, \pi^i \models \psi_1$
$M, \pi \models \psi_1 \mathbf{U} \psi_2$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models \psi_2$ and for all $0 \leq j < k$, it holds $M, \pi^j \models \psi_1$

Definition 12 (Satisfiability) Let φ be an arbitrary temporal path or state formula. Then φ is *satisfiable* iff there is a Kripke structure $M = (S, R, \mathcal{AP}, P)$ and a state $s \in S$, such that $M, s \models \varphi$. □

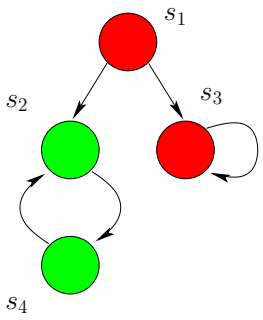
Last, we will define the Computation Tree Logic CTL$^+$, which was investigated by Johannsen and Lange in [JL03].

Definition 13 (Computation Tree Logic CTL$^+$) Let $\mathcal{AP}$ be the set of atomic propositions.

- A CTL$^+$-*state-formula* φ is a single atomic proposition $x \in \mathcal{AP}$, $\varphi \vee \varphi$, $\neg\varphi$, or $\mathbf{E}\psi$, where ψ is a CTL$^+$-*path-formula*.
- A CTL$^+$-*path-formula* ψ is a single atomic proposition $x \in \mathcal{AP}$, $\psi \vee \psi$, $\neg\psi$, $\mathbf{X}\varphi$, or $\varphi\mathbf{U}\varphi$, where φ is a CTL$^+$-state-formula.

The set CTL$^+$is defined as the CTL$^+$-state-formulas.

Remark: Informally, path formulas can only occur as subformulas of state formulas. We cannot use consecutively the temporal quantifiers without a path quantifier between



$M = (S, R, \mathcal{AP}, P)$ with
$S = \{s_1, s_2, s_3, s_4\}$,
$R = \{(s_1, s_2), (s_1, s_2), (s_3, s_3), (s_2, s_4), (s_4, s_2)\}$,
$\mathcal{AP} = \{■, ■\}$ and
$P(s_1) = ■, P(s_3) = ■, P(s_2) = ■, P(s_4) = ■$.

Figure 2.2: A Kripke structure for Example 1

them (as possible in CTL**). The main difference to* CTL* *lies in the definition of state formulas: every state formula is also a path formula.* □

In the following, we give some examples for the defined temporal logics from above.

Example 1 Referring to the Kripke structure shown in Figure 2.2, we will give some formulas that are fulfilled in M.

First, we will consider CTL*-formulas: $M, s_1 \models \mathbf{EG}■$, $M, s_2 \models \mathbf{A}■$ and $M, \pi \models ■\mathbf{U}■$ for $\pi = (s_1, s_2, s_4, s_2, s_4, \dots)$.

Then, we observe these satisfiable CTL-formulas: $M, s_1 \models \mathbf{AX}(■ \vee ■)$ and $M, s_1 \models \mathbf{EG}(\neg■ \wedge (\mathbf{AG}■))$.

Last, we give two examples for a satisfied CTL$^+$-formulas: $M, s_1 \models ■ \vee \mathbf{E}((\mathbf{X}■) \vee (■\mathbf{U}■))$ and $M, s_1 \models ■ \wedge \mathbf{E}((\mathbf{X}■) \wedge (■\mathbf{U}■))$. □

Finally, we can observe the following relation between the temporal logics:

Observation 6 CTL$\subsetneq$ CTL$^+\subsetneq$ CTL*. □

Proof Through the definition of the formulas the direction from left to right is clear. Now we show the other direction: Let $x \in \mathcal{AP}$ be an atomic proposition, then $\mathbf{XX}\neg x \in$ CTL*, but $\mathbf{XX}\neg x \notin$ CTL$^+$, and clearly $\mathbf{XX}\neg x \notin$ CTL. Let $\varphi := \mathbf{E}x$ be a formula, then $\varphi \in$ CTL$^+$, but $\varphi \notin$ CTL. ■

2.2.3 Modal Logic

As we will see in Chapter 3, some proofs will be similar to results in modal logic. Thus we will give a very short introduction into the theory of modal logics, that will help to understand these analogies. For a more elaborate introduction in this theory, we refer to the common textbook [BdRV01].

We start defining the basic *modal language* using a set of *propositional variables* $V = \{x_1, \dots, x_n\}$ for $n \in \mathbb{N}$ and the unary modal operator $\Diamond$.

Definition 14 (Modal Language) We define the modal language through the syntax of its formulas. A so-called *well-formed formula* φ is either a single propositional variable $x \in V$, its negation $\neg\varphi$, the modal constant falsum $\bot$ ("bottom", that is the same as

the logical false), a disjunction of two modal formulas $\varphi \vee \varphi$, or a modal formula prefixed by diamond $\Diamond\varphi$.

The diamond operator's opposite is the dual operator $\Box$ ("box"), which is defined by the expression $\Box\varphi := \neg\Diamond\neg\varphi$. Moreover there are the classical abbreviations for $\wedge, \rightarrow, \leftrightarrow$ and the modal constant true $\top = \neg\bot$ ("top"). □

In the following we will determine the more "philosophic" meanings of the operators $\Box$ and $\Diamond$:

- $\Diamond\varphi$ is read "it is possible that the case φ holds",
- $\Box\varphi$'s meaning is the opposite, that is "it is **not** possible that **not** the case φ holds". Much clearer is "it is necessary that the case φ holds".

Transferring these definitions to Kripke structures that are also defined in the same way in modal logic, we get the following semantics of the model property and extend the Definition 11 of the propositional logic model property

Definition 15 Let $M = (S, R, \mathcal{AP}, P)$ be a Kripke structure, φ, ψ be modal formulas and $s \in S$ be a state in M. Then the *model property* $M, s \models \varphi$ is inductively defined as follows:

- $M, \varphi \models x$ iff $x \in P(v)$.
- $M, \varphi \models \bot$ never.
- The meanings of $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi$ as usual.
- $M, v \models \Diamond\varphi$, iff there is a s' with $(s, s') \in R$ and $M, s' \models \varphi$.
- $M, v \models \Box\varphi$ iff for all s', such that $(s, s') \in R$, it holds $M, s' \models \varphi$. □

Now we can make an observation concerning the similarities between modal logic and temporal logic, that is important for Chapter 3.

Observation 7 *In the following, we show equivalences between operators in temporal logic and modal logic that rise from the semantics anchored in the Kripke structures:*

- $\Diamond$ *matches* **EX** *and*
- $\Box$ *is similar to* **AX**. □

2.3 Turing Machines and Complexity Classes

Working with decision problems in complexity theory requires an appropriate machine model to be able to classify these complexity degrees. This model will be a usual Turing machine, which will be defined below.

A *deterministic Turing machine* is a 7-tupel $M = (S, \Sigma, \Gamma, \delta, s_0, \Box, F)$, where

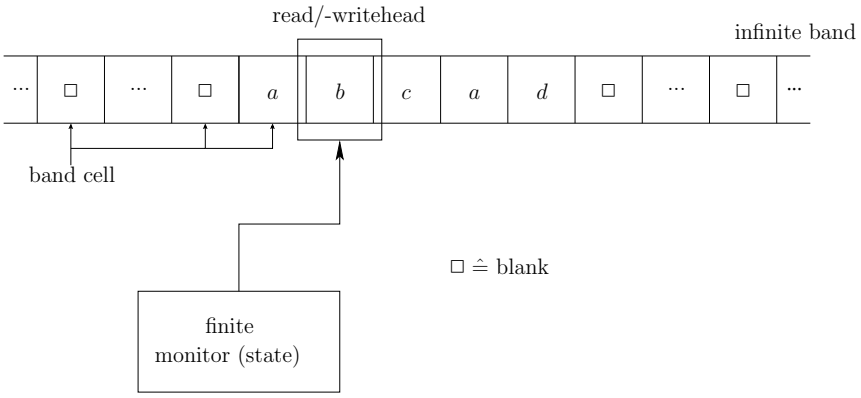


Figure 2.3: Schematic of a Turing machine

- $S := \{s_0, s_1, \ldots, s_n\}$ is the finite set of states, for $n \in \mathbb{N}$,
- Σ is the finite input alphabet,
- $\Gamma \supset \Sigma$ is the finite tape alphabet,
- $\delta \colon S \times \Gamma \to S \times \Gamma \times \{L, R, N\}$ is the transition relation,
- $\square \in \Gamma \backslash \Sigma$ is the blank symbol and
- $F \subseteq S$ is the set of accepting states.

The behavior of Turing machines is defined as usual. In Figure 2.3 the typical elements of a Turing machine are shown. For further informations concerning Turing machines please consider the textbook [Sch01]. Analogously, we define *nondeterministic Turing machines*. We assume the reader is familiar with the concept of the $\mathcal{O}$- and o-notation ("big-oh" and "little-oh"; for additional information see [Sip07, pp. 252]).

Next, we show in Table 2.1 a summarization of the relevant complexity classes in this thesis.

Table 2.1: Overview of relevant complexity classes in this thesis

TIME(t)	All sets that can be computed by some deterministic Turing machine M that is time-bounded by the function $t \colon \mathbb{N} \to \mathbb{N}$, that is, on input x the machine M needs at most $c \cdot t(\lvert x \rvert) + d$ steps until its computation finishes, for constant $c, d \in \mathbb{N}$.
NTIME(t)	All sets that can be computed by some *non*deterministic Turing machine M that is time-bounded by the function $t \colon \mathbb{N} \to \mathbb{N}$, that is, on input x the machine M needs at most $c \cdot t(\lvert x \rvert) + d$ steps until its computation finishes, for constant $c, d \in \mathbb{N}$.

continued on next page

continued from previous page

SPACE(s)	All sets that can be computed by some deterministic Turing machine M that is space-bounded by the function $s\colon \mathbb{N} \to \mathbb{N}$, that is, on input x the machine M needs at most $c \cdot s(\lvert x\rvert) + d$ cells on its working tape until its computation finishes, for constant $c, d \in \mathbb{N}$.
L	All sets that can be computed by some deterministic Turing machine in logarithmic space; equals SPACE($\log(n)$).
P	All sets that can be computed by some deterministic Turing machine in polynomial time; equals TIME$\left(n^{\mathcal{O}(1)}\right)$.
NP	All sets that can be computed by some *non*deterministic Turing machine in polynomial time; equals NTIME$\left(n^{\mathcal{O}(1)}\right)$.
PSPACE	All sets that can be computed by some deterministic Turing machine in polynomial space; equals SPACE$\left(n^{\mathcal{O}(1)}\right)$.
EXPTIME	All sets that can be computed by some deterministic Turing machine in exponential time; equals TIME$\left(2^{n^{\mathcal{O}(1)}}\right)$.
2-EXPTIME	All sets that can be computed by some deterministic Turing machine in double exponential time; equals TIME$\left(2^{2^{n^{\mathcal{O}(1)}}}\right)$.

After defining the complexity classes, we need to introduce another important concept in complexity theory: reductions. With reductions, we are able to prove a similarity between formal languages and decision problems, that is, being contained in the same complexity class. For example, given a formal language $L_1 \subseteq \Sigma^\star$ with $L_1 \in \mathrm{P}$ and another formal language $L_2 \subseteq \Gamma^\star$. If we can show, that a function $f\colon \Gamma^\star \to \Sigma^\star$ exists, and f is a total function, which is computable in polynomial time, then we achieve $L_2 \in \mathrm{P}$, too. In the following, we will strengthen these facts in a definition.

Definition 16 (Polynomial Time Many-One Reduction $\leq_m^{\mathrm{p}}$) Let $L_1 \subseteq \Sigma^\star$, and $L_2 \subseteq \Gamma^\star$ be two formal languages. L_1 is called *in polynomial time m-reducible to* B, if there exists a function $f\colon \Sigma^\star \to \Gamma^\star$ with the following two properties:

- A Turing machine M exists, that computes the output $f(x) \in \Gamma^\star$ for the given input $x \in \Sigma^\star$ in polynomial time.
- For all $x \in \Sigma^\star$, it holds $x \in L_1 \Leftrightarrow f(x) \in L_2$

We call f a polynomial reduction from L_1 to L_2, and write $L_1 \leq_m^{\mathrm{p}} L_2$. □

As usual for reductions, $\leq_m^{\mathrm{p}}$ is reflexive and transitive. Combining the definition of reductions and complexity classes, we can define two properties of formal languages, that make a result literally complete.

Definition 17 (Hardness and Completeness) Let $\mathcal{C}$ be a complexity class and let $\leq$ be a reduction. Let L_1 be a formal language.

(i) L_1 is $\mathcal{C}$-*hard*, if for all $L \in \mathcal{C}$ it holds: $L \leq L_1$.

(ii) L_1 is called to be $\mathcal{C}$-*complete under* $\leq$, if L_1 is $\mathcal{C}$-hard **and** $L_1 \in \mathcal{C}$. □

Example 2 Let $L \in \text{NP}$ be a formal language and for all languages $L' \in \text{NP}$ it holds $L' \leq_m^{\text{p}} L$. Then we call L a NP-complete language.

Let A be a formal language that is PSPACE-complete. Let $A' \notin \text{PSPACE}$ be a formal language and $A \leq_m^{\text{p}} A'$. Then A' is PSPACE-hard. □

Completeness is important, as it extends a single statement to a representative of a whole complexity class. If $\mathcal{C}$ is a complexity class, and we show for a set L its $\mathcal{C}$-completeness, then we can pull the argumentation about the whole class $\mathcal{C}$ down to the single set L. Every result for L is now a result for every member in $\mathcal{C}$. Thus completeness-results are the favorites of a theoretical computer scientist.

2.4 Clones and Post's Lattice

In this section we will introduce the mathematical part of Post's Lattice (for further informations we refer the reader to [Pip97, Chapter 1]), that are clones. First we will start to extend the definitions about Boolean functions made in Section 2.2.1.

A n-ary *Boolean function* is a function $f\colon \{0,1\}^n \to \{0,1\}$. For example, 0-ary Boolean functions are the constant functions $c_0 = 0$ and $c_1 = 1$, 1-ary Boolean functions are $id(x) = x$ (the identity) and $not(x) = \neg x$, and 2-ary Boolean functions are *and*, *or*, *xor* ($\oplus$), *eq* ($\leftrightarrow$) and *imp* ($\rightarrow$).

Let $\mathbb{B}_n$ denote the set of all n-ary Boolean functions. Let B be a set of Boolean functions. The clone $[B]$ of B is the smalltest set of Boolean functions, such that

- $B \subseteq [B]$,
- $I_k^n \in [B]$, for $I_k^n(x_1, \ldots, x_n) = x_k$ and $1 \leq k \leq n$, and
- if $g_1, \ldots, g_m \in \mathbb{B}_n \cap [B]$, $f \in \mathbb{B}_n \cap [B]$, then

  $$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), g_2(x_1, \ldots x_n), \ldots, g_m(x_1, \ldots, x_n)) \in [B],$$

  for $n, m \in \mathbb{N}$.

For every finite $B_0 \subseteq B$ with $[B_0] = B$, we call B_0 a *base* (or *basis*) for B. The whole lattice is shown in Figure 2.4, and a list of all relevant Boolean clones with bases is shown in Table 2.2.

Example 3 ([BCRV03]) Let B be a set of Boolean functions. Then $f \in [B]$ if and only if $f \in B \cup \{id\}$ or another function $g \in [B]$ and $\alpha_1, \ldots, \alpha_n$ for $n \in \mathbb{N}$, that are either variables or functions from $[B]$ exist, such that $f = g(\alpha_1, \ldots, \alpha_n)$. □

At first, we define some necessary concepts that are needed for the clones.

Definition 18 ([BCRV03]) A Boolean function f is called

- *1-reproducing* if $f(1, \ldots, 1) = 1$,
- *self-dual* if for all $a_1, \ldots, a_n \in \{0,1\}$ we have $f(a_1, \ldots, a_n) = \neg f(\overline{a_1}, \ldots, \overline{a_n})$,
- *monotonic* if for all $\alpha, \beta \in \{0,1\}^n$ holds: If $\alpha \leq \beta$ then $f(\alpha) \leq f(\beta)$.

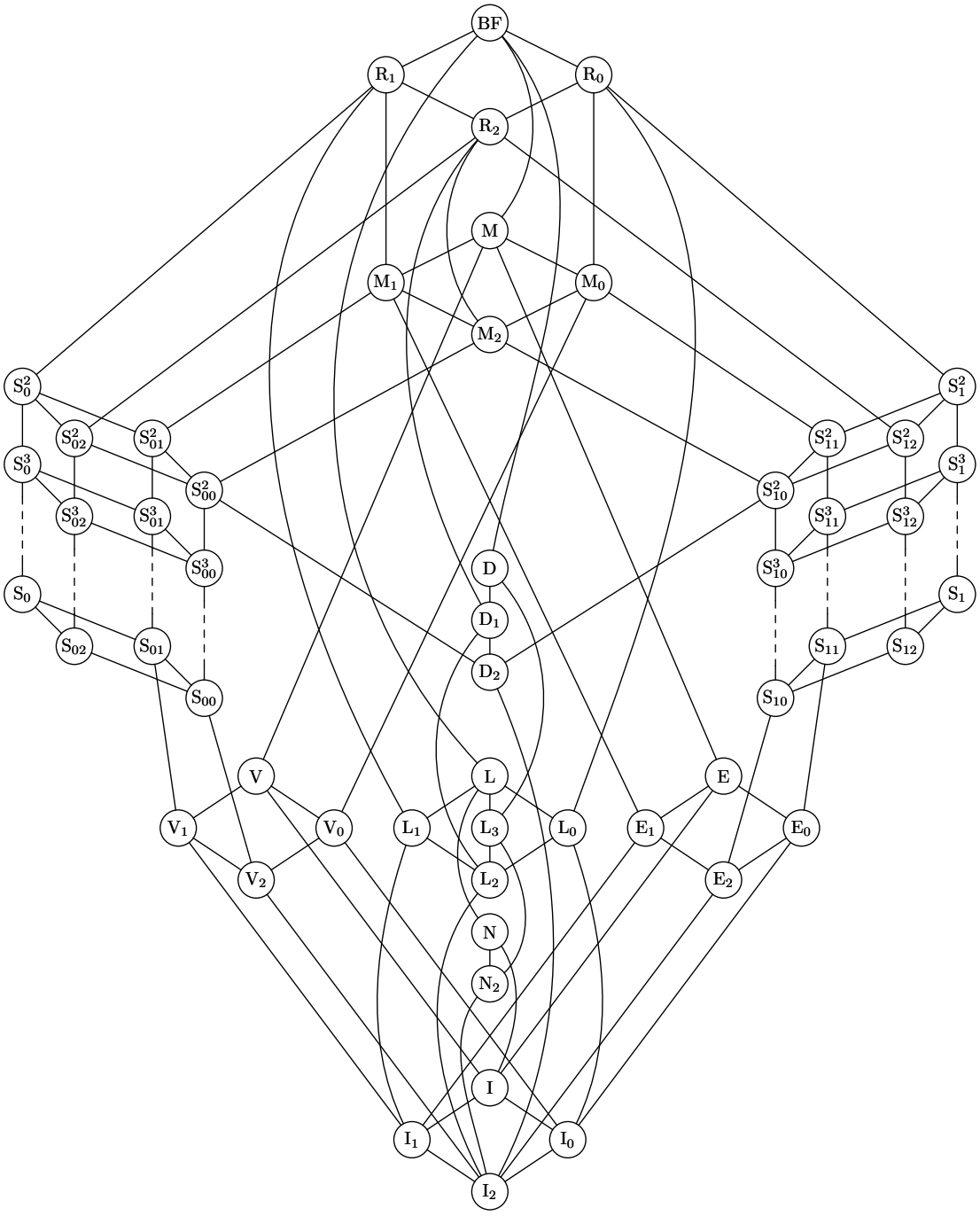


Figure 2.4: Post's Lattice (graphic by Steffen Reith [Rei])

Let $T \subseteq \{0,1\}^n$. We call T *1-separating* if an $i \in \{1,\ldots,n\}$ exists such that for all $(b_1,\ldots,b_n) \in T$ holds $b_i = 1$. A function f is called *1-separating of level k* if every $T \subseteq f^{-1}(1)$ with $|T| = k$ is 1-separating. □

In the following Table 2.2 we give a overview of the relevant clones in this thesis.

Class	**Definition**	**Base(s)**
BF	all Boolean functions	$\{\wedge, \neg\}$
R_1	$\{f \in BF \mid f$ is 1-reproducing$\}$	$\{\vee, x \oplus y \oplus 1\}$
M	$\{f \in BF \mid f$ is monotonic$\}$	$\{\wedge, \vee, 0, 1\}$
M_0	$M \cap R_0$	$\{\wedge, \vee, 0\}$
S_1	$\{f \in BF \mid f$ is 1-separating$\}$	$\{x \wedge \overline{y}\}$
S_{11}	$S_1 \cap M$	$\{x \wedge (y \vee z), 0\}$
D	$\{f \mid f$ is self-dual$\}$	$\{x\overline{y} \vee x\overline{z} \vee \overline{yz}\}$
E_2	$[\{\wedge\}]$	$\{\wedge\}$
N	$[\{\neg\}] \cup [\{0\}] \cup [\{1\}]$	$\{\neg, 1\}$, $\{\neg, 0\}$

Table 2.2: List of all relevant Boolean clones with bases.

For a complete list we refer the reader to [Pip97] or [BCRV03]. In the next example, we can see how to prove, that for a given function g the containment in another clone holds, that is $g \in [\{f\}]$.

Example 4 ([BCRV03]) Let $f(x,y) := x \wedge \overline{y}$ be a Boolean function. To prove, that the function $and(x,y)$ is in the clone of $\{f\}$, we have to find a composition of f's that is equal to $and(x,y)$. As

$$f(x, f(x,y)) = x \wedge (\overline{x \wedge \overline{y}}) = x \wedge (\overline{x} \vee y) = \underbrace{(x \wedge \overline{x})}_{\equiv 0} \vee (x \wedge y) = and(x,y),$$

we have shown that containment. For another way to solve that question is to take a look at Figure 2.4 and Table 2.2. There, we see $\{f\}$ is the base of the clone S_1. As E_2 has the base $\{and\}$ and E_2 is a subset of S_1, it follows $and \in S_1 = [\{f\}]$. □

As we now see, Post's lattice is a strong tool for working with problems, that have a restricted use of Boolean functions. It helps making statements about restricted sets of used Boolean functions, and makes it easier to argue about adjacent clones (within the lattice). If we can achieve a result for a specific clone, then we are able to expand this result to a "lower" or "upper" for its complexity degree within the lattice. Thus the result includes all subclones. In Chapter 3, the benefit we get from this structure will be much clearer to the reader.

2.5 Decision Problems

A *decision problem* is a subset of the natural numbers $\mathbb{N}$. We define the decision problem in terms of the set of inputs for which the problem returns *yes*. The problem is to decide whether a given number is in the set. Any *formal language* that can be modeled by using *Gödel numbers* is equivalent to a decision problem.

Example 5 We define the set of prime numbers: PRIME $:= \{x \mid x \text{ is prime}\}$, and the set of all words written in lower case that are a palindrome: PALINDROME $:= \{u \in \Sigma^\star \mid u = u^R\}$, where $\Sigma = \{a, b, c, \ldots, z\}$ and the operator $\cdot^R \colon \Sigma^\star \to \Sigma^\star$ mirrors the given input. PRIME and PALINDROME are decision problems. □

Decision problems in first order logic and temporal logic are essential for theoretical computer science. The first set proven to be NP-complete was in fact the Satisfiability problem SAT in propositional logic (see [Coo71]). Now, we want to give a short list of such relevant problems in our thesis shown in Table 2.3. Regarding the abbreviatory notation, "SAT" stands for "Satisfiability", "MC" denotes the "Model-Checking" problem, and $\mathcal{L} \in \{\mathrm{CTL}, \mathrm{CTL}^+, \mathrm{CTL}^\star, \mathrm{LTL}\}$.

Table 2.3: Relevant decision problems for this thesis

problem	**input and question**
SAT	*Input*: a propositional logic formula φ *Question*: Is φ satisfiable, i.e., exists an assignment θ such that $\theta \models \varphi$?
3SAT	*Input*: a propositional logic formula φ in 3CNF *Question*: Is φ satisfiable?
QBF	*Input*: a quantified Boolean formula φ *Question*: $\varphi \equiv 1$?
QBF-3SAT	*Input*: a quantified Boolean formula φ in 3CNF *Question*: $\varphi \equiv 1$?
$\mathcal{L}$-SAT	*Input*: a temporal logic formula $\varphi \in \mathcal{L}$ *Question*: Exists a Kripke structure $M = (S, R, \mathcal{AP}, P)$ and a state $s \in S$, such that $M, s \models \varphi$?
$\mathcal{L}$-SAT(A, B)	*Input*: a temporal logic formula $\varphi \in \mathcal{L}$, restricted to the temporal operators and/or quantifiers in the set A and restricted to the sentential connectives in B. *Question*: Exists a Kripke structure $M = (S, R, \mathcal{AP}, P)$ and a state $s \in S$, such that $M, s \models \varphi$?
$\mathcal{L}$-MC	*Input*: a temporal logic formula $\varphi \in \mathcal{L}$ and a Kripke structure $M = (S, R, \mathcal{AP}, P)$ *Question*: Exists a state $s \in S$, such that $M, s \models \varphi$?
$\mathcal{L}$-MC(A, B)	*Input*: a temporal logic formula $\varphi \in \mathcal{L}$, restricted to the temporal operators and/or quantifiers in the set A and restricted to the sentential connectives in B and a Kripke structure $M = (S, R, \mathcal{AP}, P)$. *Question*: Exists a state $s \in S$, such that $M, s \models \varphi$?

When talking about the satisfiability problem concerning the temporal logic $\mathcal{L}$, and we want to restrict either the temporal operators (and/or path quantifiers) or the sentential connectives, then we usually omit the unrestricted set and just write $\mathcal{L}$-SAT(C) for the restricted class C.

3 Complexity of Computation Tree Logic

In this chapter we will initially determine the complexity of the temporal logic CTL* – to be more precise the complexity of the Satisfiability problem of CTL*. As we know from [SC85] CTL*-MC is PSPACE-complete, which was proven through a two-steps-reduction from LTL-MC and LTL-SAT. First, we want to analyze fragments of CTL*-SAT. The whole set was shown to be 2-EXPTIME-hard by Vardi and Stockmeyer in [VS85]. The containment in 2-EXPTIME was shown by Emerson and Jutla in [EJ00], which yields its 2-EXPTIME-completeness.

The most common known representative of PSPACE is QBF, that we will use to show that some fragments of CTL*-SAT are PSPACE-hard. In [Sto77, pp. 12] Stockmeyer proves the PSPACE-completeness of QBF-3SAT.

3.1 Computation Tree Logic*

For CTL* we determine two kinds of results. First, we have results that yield PSPACE-hardness for the clones BF and S_1, if we restrict the operators to **A**,**E** and **X**. Furthermore we show, that CTL*-SAT is also 2-EXPTIME-complete for CTL* restricted to the clone S_1 (and all temporal and path operators). Second, we have the restricted parts of R_1 and D that can be decided in polynomial time and even in logarithmic space (restricted to the operators $\mathbf{A}, \mathbf{E}$ and $\mathbf{X}$).

3.1.1 Results for PSPACE

Formulas for QBF-3SAT have a decent form: we have a block of quantifiers that specify which assignments must fulfill the set of clauses. A set of clauses is fulfilled by a given assignment, if every single clause is fulfilled by the assignment. A single clause evaluates to *true* iff one of its literals evaluates to true. If you want to visualize this fact for a given QBF-formula, you usually model the set of all possible assignments through a complete binary tree: the root represents the first variable to which a 0 in the left subtree and a 1 in the right subtree is assigned. The two children represent the second variable and so on. At last we can see that to every leaf there is exactly one assignment, which is generated by the path to the leaf starting at the root of the tree. Therefore, we can attach the set of clauses fulfilled by the assignment to every leaf. This binary tree can be modeled by a Kripke structure and the properties we described above, can be expressed by temporal logic formula. In the following theorem, we show how this can be done.

Theorem 1 QBF-3SAT $\leq_m^{\mathrm{p}}$ CTL*-SAT. □

Proof idea Consider a formula $\varphi \in$ QBF-3SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. φ is a closed QBF of the form

$$\exists x_1 \forall x_2 \exists x_3 \ldots \exists x_{n-1} \forall x_n F$$

with F in 3CNF. Then φ is satisfiable iff all m clauses can be satisfied regarding to its quantifiers. With that in mind we take a look at the complete binary tree that corresponds to all possible assignments for $x_1, \ldots, x_n$. The quantifiers specify on which paths in the tree there must be satisfying assignments – with that, they specify which assignments (the leafs in the tree) must fulfill F.

Additionally, we use the fact that the quantifiers $\forall$ and $\exists$ correspond in the temporal world to **A** and **E**. ∎

To get a better understanding for the following proof, we encourage the reader to jump to Example 6 on page 27, at first.

Proof *The proof is inspired by [Sch02, p. 41, Satz 2.16].*

Let $\varphi \in$ QBF-3SAT be defined as above. Now we construct a formula $\mathcal{F} \in \mathrm{CTL}^\star$ of polynomial length with $\mathcal{F} \in \mathrm{CTL}^\star$-SAT if and only if $\varphi \in$ QBF-3SAT – in other words, a structure $M = (S, R, \mathcal{AP}, P)$ exists (which has the form of the complete binary tree) and a state $s \in S$ with $M, s \models \mathcal{F}$ if and only if $\varphi \equiv 1$.

We use the following abbreviated notations:

$$(\mathbf{AX})^i\psi = \begin{cases} \psi, & \text{if } i = 0 \\ \mathbf{AX}(\mathbf{AX})^{i-1}\psi, & \text{otherwise} \end{cases}$$
$$\text{and } (\mathbf{AX})^{(i)}\psi = \psi \wedge \mathbf{AX}\psi \wedge \cdots \wedge (\mathbf{AX})^i\psi.$$

Let

$$\begin{aligned} D_i &= (\mathbf{AX})^{(n)}(q_i \rightarrow (\overline{q_0} \wedge \cdots \wedge \overline{q_{i-1}} \wedge \overline{q_{i+1}} \wedge \cdots \wedge \overline{q_n})) \quad \text{for all } i = 0, \ldots, n \\ B_i &= q_i \rightarrow \Big(\mathbf{EX}\big(q_{i+1} \wedge x_{i+1} \wedge \mathcal{C}(x_{i+1})\big) \wedge \mathbf{EX}\big(q_{i+1} \wedge \overline{x_{i+1}} \wedge \mathcal{C}(\overline{x_{i+1}})\big)\Big), \\ S_i &= \Big(\big(x_i \wedge \mathcal{C}(x_i)\big) \rightarrow \mathbf{AX}\big(x_i \wedge \mathcal{C}(x_i)\big)\Big) \wedge \\ &\quad \wedge \Big(\big(\overline{x_i} \wedge \mathcal{C}(\overline{x_i})\big) \rightarrow \mathbf{AX}\big(\overline{x_i} \wedge \mathcal{C}(\overline{x_i})\big)\Big), \end{aligned}$$

for all $i = 0, \ldots, n-1$ and $\mathcal{C}\colon V_n \rightarrow \mathbb{F}$ where

$$\mathcal{C}(x) := \bigwedge_{x \in C_j} C_j \quad \text{, where } 1 \leq j \leq m.$$

D_i makes sure that in each state of the Kripke structure there is only one q_i true. B_i enforces the branching in each state of the level i to two following states in level $i+1$. Those states shall have complementary assignments with respect to the variable x_{i+1}. Additionally in each state the clauses corresponding to the literal exist. S_i passes each variable and each clause that exists in the state at level i to the next two following states in level $i+1$.

The whole formula $\mathcal{F}$ can be assembled in the following way:

$$
\begin{array}{lllllllllllllllllll}
\mathcal{F} = q_0 & \wedge & & & & & & & & & & & & & & & & & \\
 & \wedge & D_0 & \wedge & D_1 & \wedge & \dots & \wedge & D_n & \wedge & & & & & & & & & \\
 & \wedge & B_0 & \wedge & \mathbf{AX}B_1 & \wedge & (\mathbf{AX})^2 B_2 & \wedge & (\mathbf{AX})^3 B_3 & \wedge & \dots & \wedge & (\mathbf{AX})^{n-1} B_{n-1} & \wedge & & & & & \\
 & & & \wedge & \mathbf{AX}S_1 & \wedge & (\mathbf{AX})^2 S_1 & \wedge & (\mathbf{AX})^3 S_1 & \wedge & \dots & \wedge & (\mathbf{AX})^{n-1} S_1 & \wedge & & & & & \\
 & & & & & \wedge & (\mathbf{AX})^2 S_2 & \wedge & (\mathbf{AX})^3 S_2 & \wedge & \dots & \wedge & (\mathbf{AX})^{n-1} S_2 & \wedge & & & & & \\
 & \vdots & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & \wedge & (\mathbf{AX})^{n-1} S_{n-1} & \wedge & & & & & \\
 & \wedge & (\mathbf{EXAX})^{\frac{n}{2}} (C_1 \wedge \dots \wedge C_m). & & & & & & & & & & & & & & & & \\
\end{array}
$$

Claim $\varphi \in$ QBF-3SAT *iff* $\mathcal{F} \in$ CTL*-SAT. □

Proof (of Claim)

„⇒“: Let $\varphi \in$ QBF-3SAT with variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$. Now φ is satisfiable if all assignments of the variables concerning the block of quantifiers fulfill φ. That is true if all m clauses are true under all those assignments. $\mathcal{F}$ forces all Kripke structures $M = (S, R, \mathcal{AP}, P)$ to be of the form of a complete binary tree in which the root node is a state $s \in S$ with $P(s) \ni q_0$. Running through the levels to the bottom of the tree, you see which clauses are fulfilled by whichever literal. Each leaf of the tree corresponds to exactly one assignment. Because of $\mathcal{F}$'s structure, each leaf contains all fulfilled clauses by the assignment. The last term

$$(\mathbf{EXAX})^{\frac{n}{2}} (C_1 \wedge \dots \wedge C_m)$$

in $\mathcal{F}$ makes sure that each leaf, that corresponds to a must-fulfilling assignment, has to contain all m clauses. As such a Kripke structure exists, $\mathcal{F}$ is in CTL*-SAT.

„⇐“: Let $\mathcal{F} \in$ CTL*-SAT be defined as above. Let M be the fulfilling Kripke structure. Because of $\mathcal{F} \in$ CTL*-SAT the term

$$(\mathbf{EXAX})^{\frac{n}{2}} (C_1 \wedge \dots \wedge C_m)$$

is true in M. Thus each of those specified states contains $C_1, \dots, C_m$. In particular, those states contain for each variable x_i either the literal x_i or $\neg x_i$. From that we can verify that each of those assignments fulfills φ: let $\mathcal{L}$ be the set of leafs that are specified by $(\mathbf{EXAX})^{\frac{n}{2}}$, let $\lambda \in \mathcal{L}$ be one arbitrary leaf and let $l_1, \dots, l_n$ be the literals contained in λ. Because $M, s \models \mathcal{F}$ and $P(s) \ni q_0$ we know that $C_1, \dots, C_m \in \lambda$, too. Hence $C_1 \wedge \dots \wedge C_m$ will be fulfilled by the following assignment:

$$\theta(x_i) = \begin{cases} 1, & \text{if } l_i = x_i \\ 0, & \text{otherwise} \end{cases}$$

for $i = 1, \dots, n$. We can do this now for each $\lambda_i \in \mathcal{L}$, and can obtain all related assignments which are models for $C_1 \wedge \dots \wedge C_m$. Thus $\varphi \in$ QBF-3SAT. ■

Finally, we need to show that the reduction from above can be computed in polynomial time:

$$
\begin{aligned}
&|D_i| \in \mathcal{O}(n^3), \quad |B_i| \in \mathcal{O}(m), \quad |S_i| \in \mathcal{O}(m) \\
&|\mathcal{F}| = (n \cdot |D_i|)+ \\
&\qquad + (n \cdot |B_i| + \mathcal{O}(n^2))+ \\
&\qquad + (n \cdot |S_1| + n \cdot (n-1))+ \\
&\qquad + ((n-1) \cdot |S_2| + n \cdot (n-1) - 2)+ \\
&\qquad + ((n-2) \cdot |S_3| + n \cdot (n-1) - 2 - 4)+ \\
&\qquad \vdots \\
&\qquad + (|S_{n-1}| + n \cdot (n-1) - 2 - 4 - \cdots - (n-2))
\end{aligned}
$$

Thus $|\mathcal{F}| \in \mathcal{O}(|\varphi|^4)$. To compute the reduction $\langle\varphi\rangle \mapsto \langle\mathcal{F}\rangle$, we first need to analyze the number of variables and clauses in the given qBf. Then, we construct the CTL*-formula $\mathcal{F}$. Consequently, we need to analyze the fulfilled clauses in F for each possible literal (that can be done in TIME($\mathcal{O}(n \cdot |F|)$)). The size of $\mathcal{F}$ and the needed computations are bounded by polynomials, and hence can be computed in polynomial time. Furthermore, the reduction function is total, and altogether QBF-3SAT is polynomial m-reducible to CTL*-SAT, and we have QBF-3SAT $\leq_m^{\mathrm{p}}$ CTL*-SAT. ■

Now, we can observe the following fact: let φ be a closed QBF-formula, where

$$\varphi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n F,$$

and F is in CNF, and $Q_i \in \{\forall, \exists\}, 1 \leq i \leq n$. Then we need to adjust the part $(\mathbf{EXAX})^{\frac{n}{2}}(C_1 \wedge \cdots \wedge C_m)$ of the formula $\mathcal{F}$ in the following:

$$\mathcal{Q}_1 \mathcal{Q}_2 \ldots \mathcal{Q}_n (C_1 \wedge \cdots \wedge C_m),$$

where

$$\mathcal{Q}_i = \begin{cases} \mathbf{EX} & \text{, if } Q_1 = \exists \\ \mathbf{AX} & \text{, otherwise.} \end{cases}$$

With this adjustment we have a working reduction for arbitrary QBF-3SAT-formulas with an arbitrary block of quantifiers.

Summarizing the proven theorem, we have a straight reduction from the PSPACE-complete QBF-3SAT to CTL*-SAT, but not really to the whole set. Looking at the used Boolean functions, temporal operators and path quantifiers, we see that some parts of this logic are not used. This reduction leads to the fragment which is restricted to the operators **A**, **E** and **X**, and all Boolean functions (as $\wedge$ and $\neg$ occur, which are a basis for the clone BF).

As we started at QBF-3SAT, we found the first PSPACE-hard fragment for the Satisfiability problem in the logic CTL*: CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF).

To illustrate the proof to Theorem 1, we want to give a short example and show therewith how to apply the stated reduction correctly.

Example 6 Let $\varphi := \exists x_1 \forall x_2 \exists x_3 \forall x_4 \big((x_1 \vee x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_1 \vee x_2)\big)$, which is true, as

$$
\begin{aligned}
& \exists x_1 \forall x_2 \exists x_3 \forall x_4 \big((x_1 \vee x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_1 \vee x_2)\big) \\
\equiv\ & \exists x_1 \forall x_2 \exists x_3 \big((x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)\big) \wedge \\
& \exists x_1 \forall x_2 \exists x_3 (x_1 \vee x_2 \vee \neg x_3) \\
\equiv\ & \Big(\exists x_1 \forall x_2 (\neg x_1 \vee x_2) \vee \exists x_1 \forall x_2 \big((x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)\big)\Big) \wedge \\
& \Big(\underbrace{\exists x_1 \forall x_2 (1) \vee \exists x_1 \forall x_2 (x_1 \vee x_2)}_{\equiv 1}\Big) \\
\equiv\ & \big(\exists x_1 (\neg x_1) \wedge \underbrace{\exists x_1 (1)}_{\equiv 1}\big) \vee \big(\underbrace{\exists x_1 (x_1 \wedge \neg x_1) \wedge \exists x_1 (1)}_{\equiv 0}\big) \equiv (1 \vee 0) \equiv 1
\end{aligned}
$$

Now we construct the formula $\mathcal{F}$ through the method given by Theorem 1.

$$
\begin{aligned}
\mathcal{F} = {} & q_0 \wedge D_0 \wedge D_1 \wedge D_2 \wedge D_3 \wedge D_4 \wedge \\
& \wedge B_0 \wedge \mathbf{AX} B_1 \wedge \mathbf{AXAX} B_2 \wedge \mathbf{AXAXAX} B_3 \wedge \\
& \wedge \mathbf{AX} S_1 \wedge \mathbf{AXAX} S_1 \wedge \mathbf{AXAXAX} S_1 \wedge \\
& \wedge \mathbf{AXAX} S_2 \wedge \mathbf{AXAXAX} S_2 \wedge \mathbf{AXAXAX} S_3 \wedge \\
& \wedge \mathbf{EXAXEXAX} (C_1 \wedge C_2) \\
= {} & q_0 \wedge (\mathbf{AX})^{(4)} (q_0 \rightarrow (\overline{q_1} \wedge \overline{q_2} \wedge \overline{q_3} \wedge \overline{q_4})) \wedge && (3.1) \\
& \wedge (\mathbf{AX})^{(4)} (q_1 \rightarrow (\overline{q_0} \wedge \overline{q_2} \wedge \overline{q_3} \wedge \overline{q_4})) \wedge (\mathbf{AX})^{(4)} (q_2 \rightarrow (\overline{q_0} \wedge \overline{q_1} \wedge \overline{q_3} \wedge \overline{q_4})) \wedge && (3.2) \\
& \wedge (\mathbf{AX})^{(4)} (q_3 \rightarrow (\overline{q_0} \wedge \overline{q_1} \wedge \overline{q_2} \wedge \overline{q_4})) \wedge (\mathbf{AX})^{(4)} (q_4 \rightarrow (\overline{q_0} \wedge \overline{q_1} \wedge \overline{q_2} \wedge \overline{q_3})) \wedge && (3.3) \\
& \wedge \Big(q_0 \rightarrow \big(\mathbf{EX}(q_1 \wedge x_1 \wedge C_1) \wedge \mathbf{EX}(q_1 \wedge \overline{x_1} \wedge C_2)\big)\Big) \wedge && (3.4) \\
& \wedge \mathbf{AX}\Big(q_1 \rightarrow \big(\mathbf{EX}(q_2 \wedge x_2 \wedge C_1 \wedge C_2) \wedge \mathbf{EX}(q_2 \wedge \overline{x_2})\big)\Big) \wedge && (3.5) \\
& \wedge (\mathbf{AX})^2 \Big(q_2 \rightarrow \big(\mathbf{EX}(q_3 \wedge x_3) \wedge \mathbf{EX}(q_3 \wedge \overline{x_3} \wedge C_1)\big)\Big) \wedge && (3.6) \\
& \wedge (\mathbf{AX})^3 \Big(q_3 \rightarrow \big(\mathbf{EX}(q_4 \wedge x_4 \wedge C_2) \wedge \mathbf{EX}(q_4 \wedge \overline{x_4})\big)\Big) \wedge && (3.7) \\
& \wedge \mathbf{AX}\Big(\big((x_1 \wedge C_1) \rightarrow \mathbf{AX}(x_1 \wedge C_1)\big) \wedge \big((\overline{x_1} \wedge C_2) \rightarrow \mathbf{AX}(\overline{x_1} \wedge C_2)\big)\Big) \wedge && (3.8) \\
& \wedge (\mathbf{AX})^2 \Big(\big((x_1 \wedge C_1) \rightarrow \mathbf{AX}(x_1 \wedge C_1)\big) \wedge \big((\overline{x_1} \wedge C_2) \rightarrow \mathbf{AX}(\overline{x_1} \wedge C_2)\big)\Big) \wedge && (3.9) \\
& \wedge (\mathbf{AX})^3 \Big(\big((x_1 \wedge C_1) \rightarrow \mathbf{AX}(x_1 \wedge C_1)\big) \wedge \big((\overline{x_1} \wedge C_2) \rightarrow \mathbf{AX}(\overline{x_1} \wedge C_2)\big)\Big) \wedge && (3.10) \\
& \wedge (\mathbf{AX})^2 \Big(\big((x_2 \wedge C_1 \wedge C_2) \rightarrow \mathbf{AX}(x_2 \wedge C_1 \wedge C_2)\big) \wedge \big((\overline{x_2}) \rightarrow \mathbf{AX}(\overline{x_2})\big)\Big) \wedge && (3.11) \\
& \wedge (\mathbf{AX})^3 \Big(\big((x_2 \wedge C_1 \wedge C_2) \rightarrow \mathbf{AX}(x_2 \wedge C_1 \wedge C_2)\big) \wedge \big((\overline{x_2}) \rightarrow \mathbf{AX}(\overline{x_2})\big)\Big) \wedge && (3.12) \\
& \wedge (\mathbf{EXAX})^2 (C_1 \wedge C_2) && (3.13)
\end{aligned}
$$

Line (3.1) to (3.3) force for each level, that $q_1, \ldots q_4$ cannot hold simultaneously, and q_0 holds in the "first" state at level 0. (3.4) to (3.7) built the "binary tree" and (3.8) to (3.12) ensure the binding of the literals and fulfilled clauses to the associated states. Finally, (3.13) forces all clauses to be true in the appropriate state (specified by the quantification). In Figure 3.1 on page 29, we show one fulfilling Kripke structure. The

thick arrows emphasize the fulfilling paths through the structure. As we need to fulfill $(\mathbf{EXAX})^2(C_1 \wedge C_2)$ for achieving satisfiability, we chose the following path:

- first we chose the $\overline{x_1}$-way,
- then we must fulfill in both ways, whether x_2 or $\overline{x_2}$ is chosen,
- there we can chose $\overline{x_3}$, and we will always get a satisfying result (whether x_4 or $\overline{x_4}$ is used).

We have to remark, that the given Kripke structure in Figure 3.1 is simplified. Of course, there must be a successor for each "q_4"-state, what we can achieve by using a loop to itself without any loss of generality. Then, the atomic propositions are not complete for the "q_2"- and "q_3"-states – the atomic propositions from the predecessor-state must be carried over. Applying these changes, we have $M, q_0 \models \mathcal{F}$. □

In the following corollary we state the previously mentioned hardness result.

Corollary 1 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) *is* PSPACE-hard. □

Regarding Post's Lattice and the problem CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$), we have shown PSPACE-hardness for the clone BF. In our next step, we want to analyze whether this hardness result can be carried over to clones below BF. The easiest way to prove that is in stating a $\leq_m^{\mathrm{p}}$-reduction from CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) to CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, B), where B is the "new examined" clone below BF.

Theorem 2 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) $\leq_m^{\mathrm{p}}$ CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}, 1\}$). □

Proof To prove this theorem we show that each function in the base $\{\wedge, \neg\}$ can be represented with functions of the base $\{x \wedge \overline{y}, 1\}$, and these substitutions are of polynomial length. For $\neg$ we use the function

$$f_\neg(x) := 1 \wedge \overline{x},$$

and for $\wedge$ we use the function

$$f_\wedge(x, y) := x \wedge \overline{(f_\neg(y))}.$$

Those substitutions are only of constant length and they do not change the satisfiability. Thus CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{\wedge, \neg\}$) $\leq_m^{\mathrm{p}}$ CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}, 1\}$). ■

The reduction from the previous theorem yields PSPACE-hardness for that basis of Boolean functions.

Corollary 2 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}, 1\}$) *is* PSPACE-hard. □

In the following theorem, we investigate PSPACE-hardness for the clone S_1 with its basis $\{x \wedge \overline{y}\}$. We will use a very central knack, that will appear in several later proofs again.

Theorem 3 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}, 1\}$) $\leq_m^{\mathrm{p}}$ CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}\}$). □

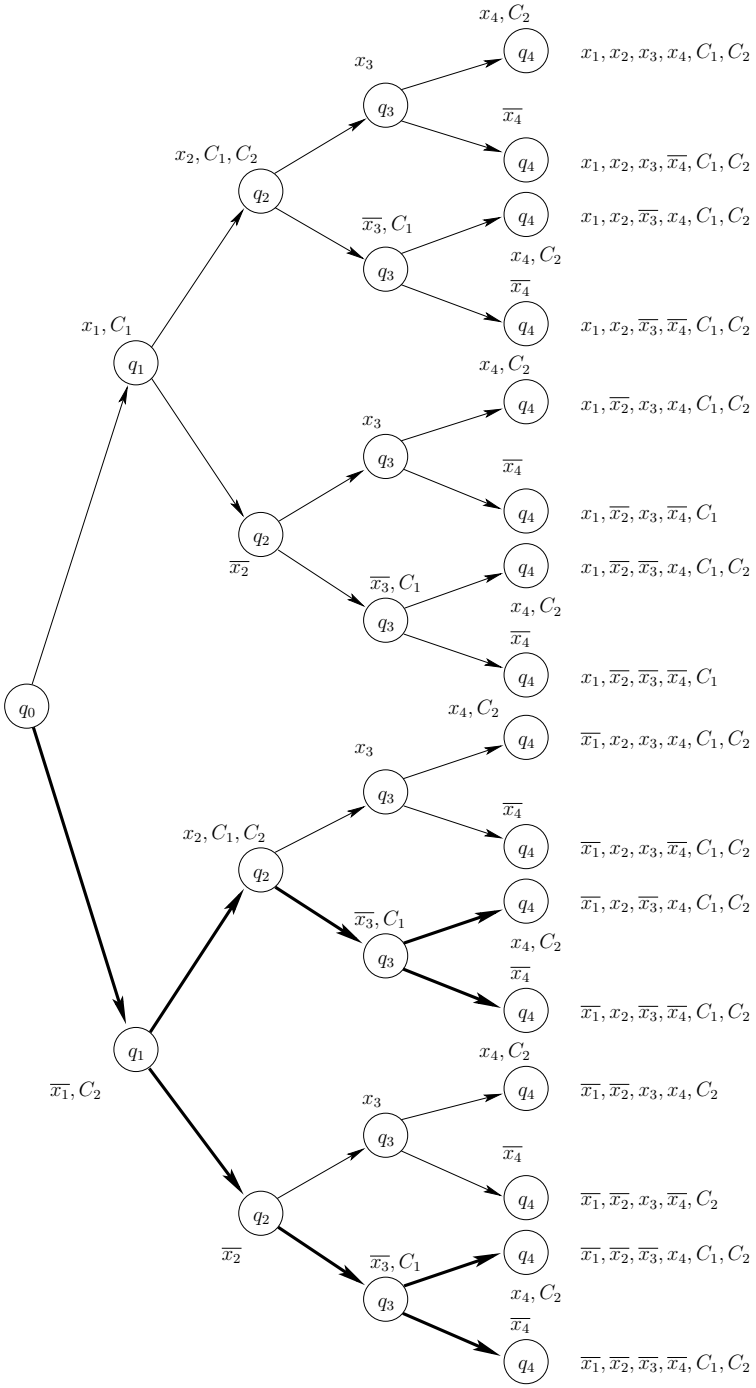


Figure 3.1: Kripke structure for Example 6

PROOF *We follow the argumentation of the proof to Theorem 3.8 in [BHSS06].*

Let $\varphi \in \text{CTL}^\star\text{-SAT}(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}, 1\})$. Let the function $d\colon \mathbb{F}_n \to \mathbb{N}$ compute the *depth* of a given CTL*-formula, which means the number nested $\mathbf{X}$s (that depth can be computed in polynomial time in $|\varphi|$). Now replace every occurence of 1 with a new variable t and force t to be *true* in every relevant state of the depending Kripke structure by adding the term of conjunctions

$$\bigwedge_{i=0}^{d(\varphi)} (\mathbf{AX})^i t.$$

This is a conjunction of linear many terms (since $d(\varphi) \leq |\varphi|$). Now, we only need to express the $\wedge$'s with the function $f_\wedge$ used in the proof to Theorem 2 and get a satisfiability-equivalent formula $\varphi' \in \text{CTL}^\star\text{-SAT}(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{x \wedge \overline{y}\})$. This reduction is total and can be computed in polynomial time. Thus we have stated the desired reduction. ■

Corollary 3 *Let B be a set of Boolean functions and $\mathrm{S}_1 \subseteq [B]$. Then* CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B$) *is* PSPACE-hard *and* CTL*-SAT(B) *is* 2-EXPTIME-complete. □

PROOF For CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B$) we can use the reduction from Theorem 3, since $\{x \wedge \overline{y}\}$ is the base of S_1. For CTL*-SAT(B), we reduce the same way as above and can keep its 2-EXPTIME-completeness. ■

Our last step for PSPACE-hard fragments will concern the clone S_{11}, but we will not be able to improve the lower complexity bound for CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$). Instead we will state a theorem that results from [Hal95], where Halpern proves an equivalent fact in the modal world.

Theorem 4 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) *for formulas with only a single atomic proposition is* PSPACE-hard. □

PROOF Let $A = Q_1 p_1 \ldots Q_m p_m A'$ be a given qBf, where $Q_i \in \{\exists, \forall\}, m \in \mathbb{N}$, and A' is a propositional formula with variables are among $p_1, \ldots, p_m$. We will now construct a formula ψ_A that is satisfiable in a Kripke structure M iff A is true. We follow the construction in our Theorem 2, and force the structure to look like a tree of truth assignments. Again, each of the leaves of the tree encodes a distinct truth assignment to the variables, and for the case that A is satisfiable, all necessary assignments should be contained in this tree.

We now need the atomic propositions $p_1, \ldots, p_m$ and $d_0, \ldots, d_{m+1}$, where the d_i denote the depth i in the tree of truth assignments. Let

$$\mathit{depth} := \bigwedge_{i=1}^{m+1} (d_i \Rightarrow d_{i-1})$$

be the formula, that simulates the relations between the d_i's.

The following formula *determined* forces the truth value of p_i is determined by depth i, and carries the value of p_i over to the proceeding states:

$$\mathit{determined} := \bigwedge_{i=1}^{m}(d_i \Rightarrow ((p_i \Rightarrow \mathbf{AX}(d_i \Rightarrow p_i)) \wedge (\neg p_i \Rightarrow \mathbf{AX}(d_i \Rightarrow \neg p_i)))).$$

Finally let $\mathit{branching}_A$ simulate the quantification of A: if Q_{i+1} is $\forall$, then each node s at depth i has two successors of depth $i+1$ (one at which p_{i+1} is true and one at which p_{i+1} is false), against what if Q_{i+1} is $\exists$, then s has at least one successor of depth $i+1$. Then $\mathit{branching}_A$ is

$$\begin{cases} ((d_i \wedge \neg d_{i+1}) \Rightarrow (\mathbf{EX}(d_{i+1} \wedge \neg d_{i+2} \wedge p_{i+1}) \wedge \mathbf{EX}(d_{i+1} \wedge \neg d_{i+2} \wedge \neg p_{i+1}))) & \text{, if } Q_{i+1} = \forall \\ ((d_i \wedge \neg d_{i+1}) \Rightarrow \mathbf{EX}(d_{i+1} \wedge \neg d_{i+2})) & \text{, if } Q_{i+1} = \exists. \end{cases}$$

Last, we can construct our final formula

$$\psi := d_0 \wedge \neg d_1 \wedge \bigwedge_{i=0}^{m}(\mathbf{AX})^i(\mathit{depth} \wedge \mathit{determined} \wedge \mathit{branching}_A \wedge (d_m \Rightarrow A')).$$

In the same way as in Theorem 2, it can be shown that ψ is satisfiable in a Kripke structure M if and only if A is true.

Next, we want to modify this reduction to a single atomic proposition. Let

$$q_j := \mathbf{EX}(\neg p \wedge (\mathbf{EX})^j p),$$

for $j \geq 1$. Let $M = (S, R, \mathcal{AP}, P)$ be a fulfilling Kripke structure. We see q_j is true at a given state s if there is a path $s_0, s_1, \ldots, s_j$, where $s_0 = s$, $(s_i, s_{i+1}) \in R$, $M, s_1 \models \neg p$, and $M, s_j \models p$. As written in [Hal95] all those formulas $q_1, q_2, \ldots$ are completely independent, and thus we can replace $p_1, \ldots, p_m, d_0, \ldots d_{m+1}$ in ψ by $q_1, q_2, \ldots, q_{2m+2}$. The resulting formula is satisfiable iff A is true, and the stated theorem applies. ■

The next step for showing PSPACE-hardness for the clone S_{11} and the problem CTL*-SAT, is finding a $\leq_m^{\mathrm{p}}$-reduction over to the clone M_0. Speaking about the modal world, this can be done as shown in [Hem05, Theorem 6.7], with proving PSPACE-hardness for formulas without atomic propositions. It is not possible to adjust this proof to the temporal world, because Kripke structures have a **total** transition relation and the knack in [Hem05, Theorem 6.7] relies on that point (where the relation must not be total). As for CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) we cannot follow that step, and hence we are not able to state a $\leq_m^{\mathrm{p}}$-reduction from CTL*($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF)-formulas without atomic propositions to CTL*($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{M}_0$)-formulas (which would be the way in modal world). Therefor we must leave this gap between CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) for formulas with one atomic proposition and CTL*($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{M}_0$). If a reduction to M_0 from BF exists, then we are able to state PSPACE-hardness for S_{11}, too. But first, we show that CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) for formulas with no atomic propositions can be decided in P.

Theorem 5 CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, *BF*) *without atomic propositions is in* P. □

PROOF Let $\varphi \in \text{CTL}^\star(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \text{BF})$ be an arbitrary temporal formula. We show that by a case-by-case induction φ can be evaluated as propositional logic formula without variables.

1) $\varphi := \mathbf{X}0$: There exists no Kripke structure $M = (S, R, \mathcal{AP}, P)$ and an infinite path π with $M, \pi \models \varphi$, because always $M, \pi^1 \not\models 0$ (logic *false* is never satisfied in a state).

2) $\varphi := \mathbf{X}1$: Every Kripke structure M and every infinite path π fulfills φ, because $M, \pi^i \models 1$ (logic *true* is satisfied in any state).

3) $\varphi := \mathbf{EX}0$ or $\varphi := \mathbf{AX}0$: Because of the same reasons as for 1) logic *false* is never satisfied in a state.

4) $\varphi := \mathbf{EX}1$ or $\varphi := \mathbf{AX}1$: Because of the same reasons as for 2) logic *true* is always satisfied in any state.

Hence we only need to take care about each 0-1-expression in the particular state. For that we only need to handle each expression stepwise which should be true in the same state. This can be done in the same way as for evaluating a complete logical expression over the Boolean functions in BF and the constants 1 and 0. In other words, we only evaluate a propositional logic formula with an assignment applied to it (and do that stepwise). Obviously this can be done in polynomial time. Thus CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) without atomic propositions is in P. ■

In the following we will give some examples to illustrate the proof for the previous theorem.

Example 7 With "=" we denote an equal transformation (like known from the propositional logic, e.g., $0 \vee 1 = 1$).

(a) $\mathbf{EX}((\mathbf{EXX}0) \vee (\mathbf{EX}1)) \vee (\mathbf{EXEXX}1) = \mathbf{EX}(0 \vee 1) \vee 1 = \mathbf{EX}(1) \vee 1$ which is satisfiable in any Kripke structure.

(b) There exists no satisfying Kripke structure for $\mathbf{A}0 \wedge \psi$, where ψ is an arbitrary formula without atomic propositions in $\text{CTL}^\star(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \text{BF})$. □

If it can be shown, that we can also prove PSPACE-hardness, then we are able to state the collapse of the polynomial time hierarchy.

Corollary 4 *If* CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, *BF*) *without atomic propositions is* PSPACE-*hard, then* P = PSPACE. □

PROOF Let CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) without atomic propositions be PSPACE-hard. As shown in Theorem 5, CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) is in P. With its PSPACE-hardness we are able to reduce all problems in PSPACE within polynomial time to the problem CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$, BF) without atomic propositions, and decide it through the stated P-algorithm. Hence all problems in PSPACE can be solved in polynomial time, and thus P = PSPACE. ■

Now we show that a $\leq_m^{\mathrm{p}}$-reduction from M_0 to S_{11} exists. This can be helpful to show a possible PSPACE-hardness for S_{11} with arguing about M_0.

Theorem 6 CTL*-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{\wedge, \vee, 0\}) \leq_m^{\mathrm{p}}$ CTL*-SAT$(\{\boldsymbol{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{S}_{11})$. □

Proof Let $\varphi \in$ CTL*-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \{\wedge, \vee, 0\})$. As in Theorem 2 we need to express the base $\{\wedge, \vee, 0\}$ with the base of $\mathrm{S}_{11} = \{x \wedge (y \vee z), 0\}$. But at first, we do this with $[\mathrm{S}_{11} \cup \{1\}] = M$:

$$f_\wedge(x, y) := x \wedge (y \vee 0)$$
$$f_\vee(x, y) := 1 \wedge (x \vee y)$$

and for 0 we do not need any substitution as $0 \in S_{11}$. By now, the given functions are in $[\mathrm{S}_{11} \cup \{1\}] = \mathrm{M}$. To achieve S_{11} we use the same knack as in the proof to Theorem 3 and replace each occurrence of 1 with a new variable t which is set to *true* in all states by adding the same conjunction

$$\bigwedge_{i=0}^{d(\varphi)} (\mathbf{AX})^i t.$$

As stated in the proof to Theorem 3 the given reduction can be computed in polynomial time. Therefore the theorem applies. ■

Recapitulating the shown results for CTL*-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\})$, we have PSPACE-hardness for the clones $\mathrm{BF}, \mathrm{R}_0$ to S_1, and showed the problem for S_{11} is of the same degree as M_0. Then we stated in Theorem 5 that for each CTL*-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\})$-formula without any atomic propositions, it can be decided in polynomial time whether a fulfilling Kripke structure exists or not. The PSPACE-hard results are illustrated with red colored circles in Figure 3.2.

Furthermore, we were able to pull the 2-EXPTIME-completeness bound for CTL*-SAT down to the clone S_1.

3.1.2 Results for P and L

In this subsection, we investigate formulas for CTL*, that will only be restricted to Boolean functions – all temporal operators and path quantifiers will stay available. In the next theorem we will examine the complexity of CTL* restricted to Boolean functions of the clone R_1.

Theorem 7 *Let B be a finite set of Boolean functions such that $[B] \subseteq \mathrm{R}_1$. Then* CTL*-SAT$(B)$ *is in* P. □

Proof Let $\varphi \in$ CTL* be a formula restricted to the Boolean functions of $\mathrm{R}_1 = [\{\vee, x \oplus y \oplus 1\}]$, and let $\mathcal{AP}$ be the set of atomic propositions. We will show that a Kripke structure M which is a model for φ exists for every such formula. We define $M = (S, R, \mathcal{AP}, P)$ as follows:

$$S := \{s\}$$
$$R := \{(s, s)\}$$
$$P(s) := \mathcal{AP}.$$

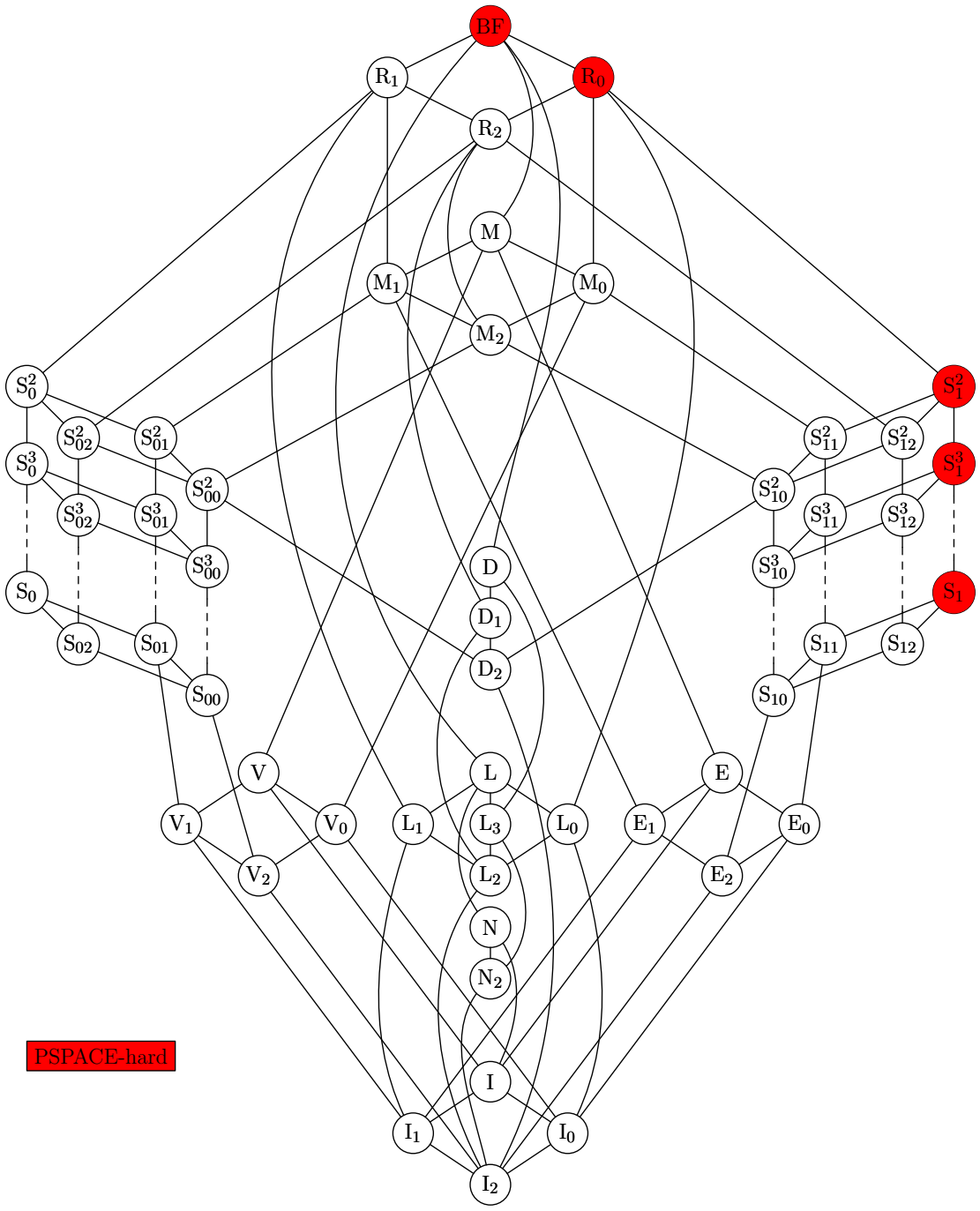
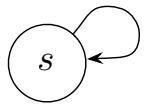


Figure 3.2: PSPACE-hard fragments for CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)



$P(s) = \mathcal{AP}$

Figure 3.3: Kripke structure M for Theorem 7

M is visualized in Figure 3.3. Now we prove by induction on the length of φ that $M, s \models \varphi$.

Initial Step. $\varphi = x$ and $x \in \mathcal{AP}$ is an atomic proposition. Then $M, s \models \varphi$ is true, because $x \in P(s)$.

Inductive Steps. Let $\varphi, \psi \in \mathrm{CTL}^\star(\mathrm{R}_1)$ be formulas, let x be a new atomic proposition. Let $u = (s, s, \dots)$ be a path in M. Then

- $\varphi \vee x$ is true because each atomic proposition of φ is in $P(s)$ and $x \in P(s)$, too,
- the same holds for $\varphi \oplus x \oplus 1$, because $\varphi \oplus x \equiv 0$ and $0 \oplus 1 \equiv 1$,
- $M, u \models \mathbf{X}\varphi$ is always true, because the atomic propositions of φ are in $P(s)$ and $(s, s) \in R$,
- $M, u \models \varphi \mathbf{U} \psi$ is always true, because for any $k \geq 0$ the following holds in a trivial way (atomic propositions of φ, ψ are in $P(s)$)

$$M, u^k \models \psi \text{ and for each } 0 \leq j < k : M, u^j \models \varphi,$$

- $M, u \models \mathbf{F}\varphi$ is always true, because all atomic propositions from φ are in $P(s)$,
- $M, u \models \mathbf{G}\varphi$ is always true, because φ holds in each state on the path u,
- $M, s \models \mathbf{A}\varphi$ is always true, because of the s-loop and
- $M, s \models \mathbf{E}\varphi$ is true, as $M, u \models \varphi$.

Thus every arbitrary formula $\varphi \in \mathrm{CTL}^\star(R_1)$ is also in CTL*-SAT which can be decided in polynomial time. According to the precondition $[B]$ is a subset of R_1. Hence we only need to check whether we have 1-reproducing functions or not, which can be done in polynomial time. Hence we have proven the stated theorem. ■

Corollary 5 *Let B be a finite set of Boolean functions such that $[B] \subseteq \mathrm{R}_1$. Then* CTL*-SAT($B$) *is in* L. □

Proof As stated in Theorem 7 all CTL*(R1)-formulas, and all CTL*(B)-formulas with $[B] \subseteq \mathrm{R}_1$ are satisfiable. Hence we only need to check the syntactical correctness of the formula, which can be done in L. ■

As we can see now, the question whether a given temporal logic formula in CTL* restricted to functions of clone R_1 (or any other clone below R_1) is satisfiable, is very easy to decide, i.e., in logarithmic space. As R_1 is nearly on the top of Post's Lattice (it

is one of the clones that are directly below the clone of all Boolean functions BF), a very huge part of the lattice is included in R_1, and hence as easy to decide.

Further, we will show a similar result for the clone D. Besides, the proof for the theorem has its roots in the world of modal logic.

Theorem 8 *Let B be a finite set of Boolean functions such that $[B] \subseteq \mathrm{D}$. Then* CTL*-SAT(B) *is in* P. □

Proof idea *We follow the proof of Lemma 3.13 in [BHSS06].*

As D contains all self-dual functions ($f(a_1, \ldots, a_n) = \neg f(\overline{a_1}, \ldots, \overline{a_n})$), we can easy find a Kripke structure that is a model for a given formula $\varphi \in \mathrm{CTL}^\star(\mathrm{D})$. As in Theorem 7, the main idea is to work with a Kripke structure that consists of a single state with only a looping edge to itself. First, we start labeling all atomic propositions to the single state. If we do not get satisfiability by now, we just remove all propositions from the state. Since φ only contains self-dual functions the now obtained labeling in the Kripke structure transforms to a satisfying model. ■

Proof Let $M = (S, R, P)$ be the Kripke structure of the proof of Theorem 7. Thus $P(s) = \mathcal{AP}$. Let $\overline{M} = (S, R, \overline{P})$ the same Kripke structure but $\overline{P}(s) = \emptyset$. We prove by induction on the length of φ that $M, s \models \varphi$ iff $\overline{M}, s \not\models \varphi$.

Initial Step. Let $\varphi = x$ for a single atomic proposition x. Then $M, s \models \varphi$ because $x \in P(s)$, and hence $\overline{M}, s \not\models \varphi$ and $M, s \models \varphi$.

Inductive Steps. Let $M, q \models \varphi$ and let $u = (s, s, \ldots)$ be a path in M and $\overline{M}$, respectively. Let $M, s \models \psi$. Then

- $\mathbf{A}\varphi, \mathbf{E}\varphi, \mathbf{X}\varphi \in \mathrm{CTL}^\star\text{-SAT}(D)$, since by induction φ is satisfied in M, s iff φ is false in $\overline{M}, s$.
- $M, u \models \varphi \mathbf{U} \psi$ because for any $k \geq 0$ the following holds due to the s-loop:

$$M, u^k \models \psi \text{ and for each } 0 \leq j < k : M, u^j \models \varphi$$
$$\textit{iff}$$
$$\overline{M}, u^k \not\models \psi \text{ and for each } 0 \leq j < k : \overline{M}, u^j \not\models \varphi.$$

- $\mathbf{F}\varphi, \mathbf{G}\varphi \in \mathrm{CTL}^\star\text{-SAT}(D)$ because either all atomic propositions are in $P(s)$ (thus they are satisfied in M, s) or none (therefore they are satisfied in $\overline{M}, s$).
- Now let $\psi = g(\varphi_1, \ldots, \varphi_n)$ for $g \in D$ and $\varphi_1, \ldots, \varphi_n \in \mathrm{CTL}^\star\text{-SAT}(D)$. Since $g \in D$ it holds that $M, s \models g(\varphi_1, \ldots, \varphi_n)$ iff $M, s \models \neg g(\neg\varphi_1, \ldots, \neg\varphi_n)$. By induction, $M, s \models \neg g(\neg\varphi_1, \ldots, \neg\varphi_n)$ is equivalent to $\overline{M}, s \models \neg g(\varphi_1, \ldots, \varphi_n)$ which is the same as $\overline{M}, s \not\models g(\varphi_1, \ldots, \varphi_n)$.

Thus we just need to check whether $M, s \models \varphi$ or $\overline{M}, s \models \varphi$ for a given CTL$^\star(D)$-formula to obtain a satisfying structure. This test can be done in polynomial time.

As mentioned in the Theorem 7, our precondition states $[B]$ is a subset of D, and with that, we have shown the containment in P. ■

Corollary 6 *Let B be a finite set of Boolean functions such that $[B] \subseteq \mathrm{D}$. Then* CTL*-SAT(B) *is in* L. □

PROOF Every formula $\varphi \in \mathrm{CTL}^\star(\mathrm{D})$, and in particular every formula in $\mathrm{CTL}^\star(B)$ with $[B] \subseteq \mathrm{D}$ is satisfiable as we see in the proof by induction in Theorem 8. Thus we only need to check the correct syntax of φ which is in L. ■

In conclusion of this subsection, we have shown that huge parts of Post's Lattice are very easy to decide regarding the temporal logic $\mathrm{CTL}^\star$, because R_1 is on the top of the lattice and has many subclone-elements.

This is very interesting, as for R_0 we got a PSPACE-hardness result in contrast to L for R_1. Concerning the previous section, we complete the illustration of our results of Figure 3.2 in the new Figure 3.4 on page 38, where the green circles denote the complexity degree L.

3.2 Computation Tree Logic

In this section we examine the more restricted branching time logic CTL. We will transfer the results from Section 3.1 to this logic and expand them to a completeness result. First, we start with a section about the PSPACE-results, and will prove the containment in PSPACE for the logic restricted to the CTL-quantifiers **AX** and **EX**. Later on, we observe similar results to the Section 3.1.2 for the classes P and L in our second subsection.

3.2.1 Results for PSPACE

Theorem 9 $\mathrm{CTL\text{-}SAT}(\{\mathbf{AX}, \mathbf{EX}\}, \mathrm{BF})$ *is in* PSPACE. □

PROOF To prove this theorem we will state an algorithm whose needed space is bounded by a polynomial. Therefore, we adapt the algorithm K-WORLD from [Lad77, Theorem 5.1] to the temporal world.

At first we will define a recursive procedure STATECHECK (see Algorithm 3.1 one page 39) which operates with the four parameters $\mathscr{T}, \mathscr{F}, \tilde{\mathscr{T}}$ and $\tilde{\mathscr{F}}$ that will evolve into sets of atomic propositions. The procedure will return *true* if a Kripke structure $(S, R, \mathcal{AP}, P)$ and a state $s \in S$ exist, such that

$$M, s \models \bigwedge_{\varphi \in \mathscr{T}} \varphi \wedge \bigwedge_{\varphi \in \mathscr{F}} \neg\varphi \wedge \bigwedge_{\varphi \in \tilde{\mathscr{T}}} \mathbf{AX}\varphi \wedge \bigwedge_{\varphi \in \tilde{\mathscr{F}}} \neg\mathbf{AX}\varphi$$

is *true*. Informally, all atomic propositions in $\mathscr{T}$ have to be *true* in the specific state s and all atomic propositions in $\mathscr{F}$ have to be *false* in s. Then each accessible state s' from s (hence $(s, s') \in R$) must fulfill the members from $\tilde{\mathscr{T}}$ and a reachable state exists that does not include the propositions in $\tilde{\mathscr{F}}$.

We assume without any loss of generality the formula does not contain any **E**'s ($\neg\mathbf{A}\neg f \equiv \mathbf{E}f$) or $\vee$'s ($g \vee f \equiv \neg(\neg g \wedge \neg f)$). Thus we only need to check the **AX**- and $\wedge$-case.

The main idea of Algorithm 3.1 on page 39 is a level-based search for contradictory propositions. The sets $\mathscr{T}$ and $\mathscr{F}$ must be disjunct, and the same for the global sets $\tilde{\mathscr{T}}$ and $\tilde{\mathscr{F}}$. First, the algorithm starts to extract the atomic propositions for a given

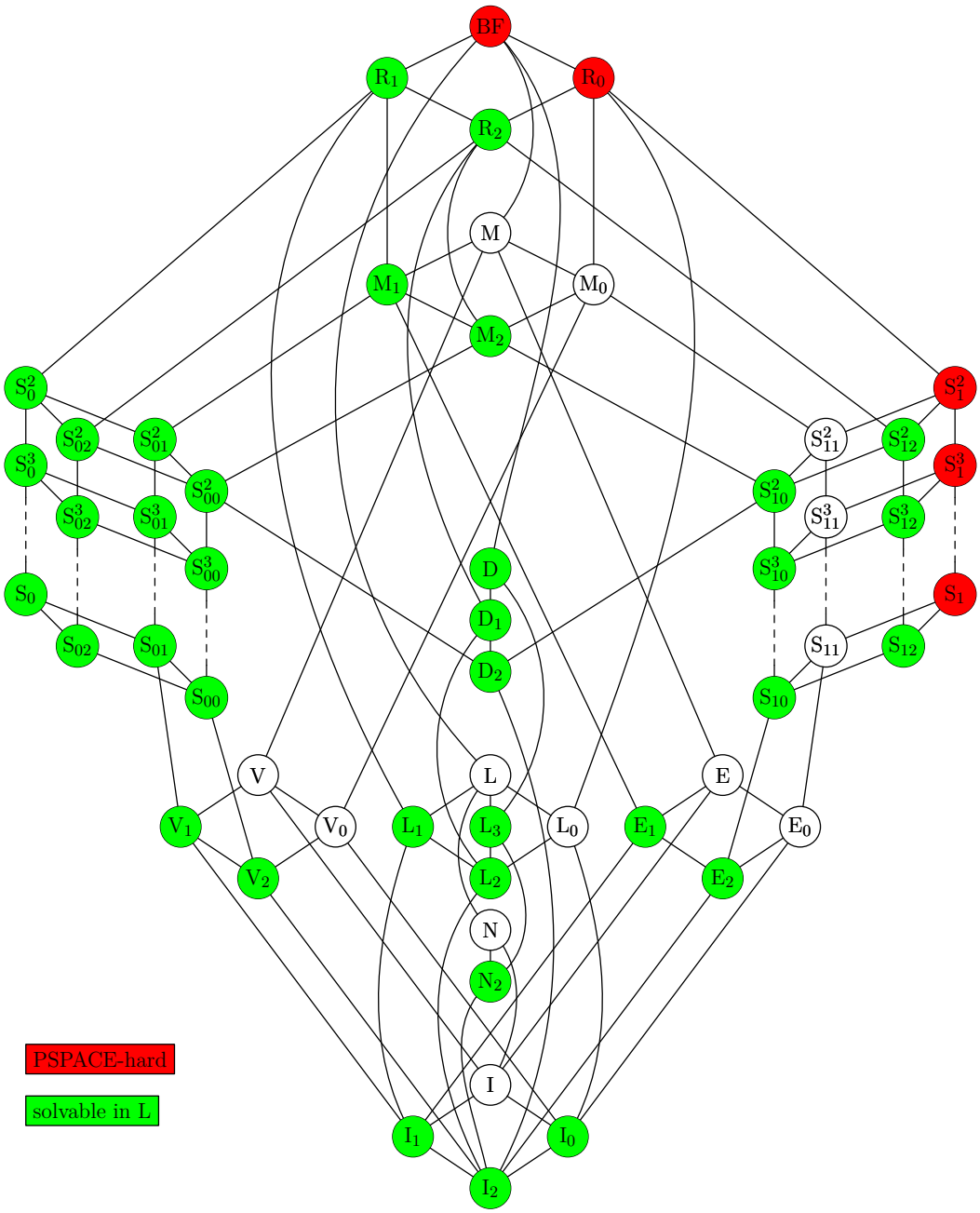


Figure 3.4: PSPACE-hard fragments for CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$), and clones in L

Algorithm 3.1 StateCheck($\mathscr{T}, \mathscr{F}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$)

if $\mathscr{T} \cup \mathscr{F} \not\subseteq \mathcal{AP}$ **then**
  Choose $\alpha \in \{\mathscr{T} \cup \mathscr{F}\} \backslash \mathcal{AP}$
  if $\alpha = \neg\beta$ **and** $\alpha \in \mathscr{T}$ **then**
    return StateCheck($\mathscr{T} \backslash \{\alpha\}, \mathscr{F} \cup \{\beta\}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$);
  else if $\alpha = \neg\beta$ **and** $\alpha \in \mathscr{F}$ **then**
    return StateCheck($\mathscr{T} \cup \{\beta\}, \mathscr{F} \backslash \{\alpha\}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$);
  else if $\alpha = \beta \wedge \gamma$ **and** $\alpha \in \mathscr{T}$ **then**
    return StateCheck($(\mathscr{T} \cup \{\beta, \gamma\}) \backslash \{\alpha\}, \mathscr{F}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$);
  else if $\alpha = \beta \wedge \gamma$ **and** $\alpha \in \mathscr{F}$ **then**
    return StateCheck($\mathscr{T}, (\mathscr{F} \cup \{\beta\}) \backslash \{\alpha\}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$)$\vee$
    StateCheck($\mathscr{T}, (\mathscr{F} \cup \{\gamma\}) \backslash \{\alpha\}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}}$);
  else if $\alpha = \mathbf{AX}\beta$ **and** $\alpha \in \mathscr{T}$ **then**
    return StateCheck($\mathscr{T} \backslash \{\alpha\}, \mathscr{F}, \tilde{\mathscr{T}} \cup \beta, \tilde{\mathscr{F}}$);
  else if $\alpha = \mathbf{AX}\beta$ **and** $\alpha \in \mathscr{F}$ **then**
    return StateCheck($\mathscr{T}, \mathscr{F} \backslash \{\alpha\}, \tilde{\mathscr{T}}, \tilde{\mathscr{F}} \cup \beta$);
  end if
end if
if $\mathscr{T} \cup \mathscr{F} \subseteq \mathcal{AP}$ **then**
  if $\mathscr{T} \cap \mathscr{F} \neq \emptyset$ **then**
    return *false*;
  else
    return $\bigwedge_{\beta \in \tilde{\mathscr{F}}}$ StateCheck($\tilde{\mathscr{T}}, \{\beta\}, \emptyset, \emptyset$)
  end if
end if

formula and puts the received propositions in their respective sets. Stepwise until all propositions are checked for the actual state, the global propositions are inspected as if they belong to one single state. If there are none contradictions, then a fulfilling Kripke structure exists for the investigated formula $\varphi \in \mathrm{CTL}(\{\mathbf{AX}, \mathbf{EX}\}, \mathrm{BF})$ and the algorithm returns *true*.

The underlying test of satisfiability is the following procedure-call, described in Algorithm 3.2.

Algorithm 3.2 CTL-AEX-SAT

Require: $\varphi \in \mathrm{CTL}(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{BF})$;
1: *value* $\leftarrow$ StateCheck$(\{\varphi\}, \emptyset, \emptyset, \emptyset)$;
2: **return** *value*;

Finally, we determine the space complexity of the Algorithm 3.1. In each recursive call we need to carry over the elements of each set. Therefore we use pointers for each subformula – this can be implemented by special type of marks for each of the four subsets. Thus the storage in each recursive step is bounded by $\mathcal{O}(n)$ related to the given input length of $|\varphi| = n$.

Thereafter, we need to examine the depth of the recursion. Let $\mathscr{S}$ be a finite set of formulas and define $|\mathscr{S}| = \sum_{\alpha \in \mathscr{S}} |\alpha|$. Now, we will prove by induction over $|\varphi| = n$ that the depth of the recursion is bounded by $2 \cdot n$.

Initial step. Let $|\varphi| = 1$. Now $\mathscr{T} \cup \mathscr{F} \subseteq \mathcal{AP}$ and let the first recursive call be denoted by the upper single quotation „′“. Then $|\mathscr{T}'| + |\mathscr{F}'| + |\tilde{\mathscr{T}}'| + |\tilde{\mathscr{F}}'| = 1 \leq 2$.

Inductive Steps. Assume the result for all numbers $< n$. Then we have to examine two cases.

(i) $\mathscr{T} \cup \mathscr{F} \neq \emptyset$: Then, in each possible recursive call $|\mathscr{T}'| + |\mathscr{F}'| + |\tilde{\mathscr{T}}'| + |\tilde{\mathscr{F}}'| < n$ (at least) since in every call the concerning subformula decreases by at least 1 (in lines 1–16).

(ii) $\mathscr{T} \cup \mathscr{F} = \emptyset$: Thus we can only be in line 21 for a recursive call. Since always $\mathscr{F}' \neq \emptyset$ we can follow case (i).

Altogether we have a reduction of $|\alpha| = |\mathscr{T}| + |\mathscr{F}| + |\tilde{\mathscr{T}}| + |\tilde{\mathscr{F}}|$ every two calls by at least 1. Thus the algorithm StateCheck has a recursion depth of $\leq 2 \cdot |\alpha| \leq 2 \cdot n$. Hence the space complexity is $\mathrm{SPACE}(\mathcal{O}(n^2)) \in \mathrm{PSPACE}$. ■

Now we will examine the results from the Section 3.1. In the following corollary, we summarize the relevant ones for PSPACE.

Corollary 7 *Let B be a set of Boolean functions and let* $\mathrm{S}_1 \subseteq [B]$. *Then* CTL-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B)$ *is* PSPACE-hard. □

PROOF The reductions in Theorem 3 on page 30 are only substitutions within the used Boolean functions. Thus we can use them for CTL, too, and will stay in this logic. ■

Finally, we will expand the obtained and adapted results to a completeness result for PSPACE.

Corollary 8 *Let B be a set of Boolean functions and let* $\mathrm{S}_1 \subseteq [B]$. *Then* CTL-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B$) *is* PSPACE-complete. □

Proof In Corollary 7, we have shown the containment in PSPACE, and in Theorem 9, we have shown the hardness for PSPACE. These facts together, we have a completeness result for PSPACE. ■

Corollary 9 *Let B be a set of Boolean functions such that* $\mathrm{S}_1 \subseteq [B]$. *Then* CTL-SAT(B) *is* EXPTIME-complete. □

Proof We use the same reduction as in Theorem 2. ■

In conclusion, we could show for CTL-SAT its containment in PSPACE and hence could state a completeness result for the class PSPACE. Herewith, this followed for every clone above S_1.

3.2.2 Results for P and L

Finally, we transfer the L-results from the Section 3.1 to the CTL-case and summarize the relevant ones in the following corollary.

Corollary 10 *Let B be a set of Boolean functions.*

(1) Let $[B] \subseteq \mathrm{R}_1$. *Then* CTL-SAT($B$) *is in* L.

(2) Let $[B] \subseteq \mathrm{D}$. *Then* CTL-SAT(B) *is in* L. □

Proof All CTL-formulas with only functions of R_1 or D are satisfiable in the same way as or CTL*. ■

Thereafter, we determine the complexity for the clones N and I restricted to the operators **AX** and **EX**.

Theorem 10 *Let B be a set of Boolean functions such that* $[B] \subseteq \mathrm{N}$. *Then* CTL-SAT($\{\mathbf{AX}, \mathbf{EX}\}, B$) *is in* L. □

Proof idea In the following proof, we will consider formulas restricted to the base $\{\neg, 0, 1\}$, as we can easily reduce it to the base $\{\neg, 0\}$ or $\{\neg, 1\}$, which are bases of N.

Such formulas have a certain look: starting with arbitrary combinations of **AX**'s and **EX**s, they contain either a single literal or a single constant. In the first case, we can always construct a Kripke structure that is a model for the formula, as we can add or remove the label of the belonging variable. In the second case, we have to examine the two different cases "1" and "0". For "1" we can built such a fulfilling structure. As the transistion relation in Kripke structures must be total, we always need a successor for a specific state. Hence we cannot construct a fulfilling structure for the case "0" – even not for the case we have the operator **AX** within the preceding block of quantifiers.

Hence we only need to analyze a given formula for these characteristics, to know whether a Kripke structure exists or not, i.e., whether $\varphi \in$ CTL-SAT or not. ■

PROOF Let $\varphi \in \text{CTL}(\{\mathbf{AX}, \mathbf{EX}\}, \{\neg, 1, 0\})$ be a temporal CTL-formula restricted to the operators $\mathbf{AX}, \mathbf{EX}$ and sentential connectives of $\{\neg, 1, 0\}$. As we only have unary functions in this base, we can transform φ to

$$\varphi' := O_1 O_2 \dots O_k \lambda,$$

where $O_i \in \{\mathbf{AX}, \mathbf{EX}\}$ for $1 \leq i \leq k, k \in \mathbb{N}$ and $\lambda \in \{0, 1, x, \neg x\}$ for an atomic proposition x.

If $\lambda \in \{1, x, \neg x\}$ for an atomic proposition x, we will show by induction over the number of preceding operators O_i, that a Kripke structure $M = (S, R, \mathcal{AP}, P)$ and a state $s \in S$ exist, such that $M, s \models \varphi'$.

Initial Step. $k = 0$: If $\lambda = x$, then M has only one state $s_0 \in S, \mathcal{AP} = \{x\}, (s_0, s_0) \in R$ and $P(s_0) = x$. Thus $M, s_0 \models \varphi'$. If $\lambda = \neg x$, then we use $P(s_0) = \emptyset$ instead. Again, it holds $M, s_0 \models \varphi'$. If $\lambda = 1$ then obviously $M, s_0 \models \varphi'$. Thus the formula $\varphi' \in \text{CTL-SAT}(\{\mathbf{AX}, \mathbf{EX}\}, \{\neg, 1, 0\})$.

Inductive Steps. $n \to n + 1$: Assume $M, s \models \varphi'$ with n preceding operators and for a state s_0. Then we obtain a Kripke structure M' from M by adding (s', s_0) to R and adding s' to S. Thus $M', s' \models \mathbf{EX}\varphi'$ and $M', s' \models \mathbf{AX}\varphi'$.

Hence the new formula is in $\text{CTL-SAT}(\{\mathbf{AX}, \mathbf{EX}\}, \{\neg, 1, 0\})$, and therefore the inductive proposition holds.

As $M, s \not\models \mathit{false}$ for any Kripke structure $M = (S, R, \mathcal{AP}, P)$ and any state $s \in S$, the formula φ' is not satisfiable if and only if $\lambda = 0$.

Thus we only need to check syntactical correctness (which can be done in L) and transform a given formula φ into the form from above (which can be done in L, as we only need to move the negations inwards) to check whether $\varphi \in \text{CTL-SAT}(\{\mathbf{AX}, \mathbf{EX}\}, \{\neg, 1, 0\})$ or not. Subsuming, this can be done in logarithmic space. As the base $\{\neg, 1, 0\}$ can be easily reduced to N, hence the stated theorem applies. ■

In the following theorem we will show, that we get a similar result if we do not make any restrictions to the CTL-operators.

Theorem 11 *Let B be a set of Boolean functions and let $[B] \subseteq \text{N}$. Then* $\text{CTL-SAT}(B)$ *is in* L. □

PROOF We will expand the proof by induction for Theorem 10 to prove this theorem. The main idea for the CTL-operators containing **U**, is to use the Kripke structure for the second formula, as it is not necessary to satisfy the first **U**-parameter.

Let $\varphi, \psi \in \text{CTL}(N)$ with n preceding Operators, M and M' be Kripke structures and s, s' be states, such that $M, s \models \varphi$ and $M', s' \models \psi$. Now we will show the remaining *inductive steps*:

- $\tilde{M}, \tilde{s} \models \mathbf{AF}\varphi$ for the Kripke structure $\tilde{M} = (\tilde{S}, \tilde{R}, \mathcal{AP}, P)$ with $\tilde{S} := S \cup \{\tilde{s}\}$ and $\tilde{R} := R \cup \{(\tilde{s}, s)\}$ with $\tilde{s} \notin S$ as a new state not contained in S.
- $M, s \models \mathbf{EF}\varphi$ as $M, s \models \varphi$.
- For $\mathbf{AG}\varphi$ and $\mathbf{EG}\varphi$ we can modify M to $\hat{M}$, such that $\hat{M}, s \models \mathbf{EG}\varphi$ and $\hat{M}, s \models \mathbf{AG}\varphi$. For that we only need to "copy" M $|S|$-times and add an edge (r, s) for each

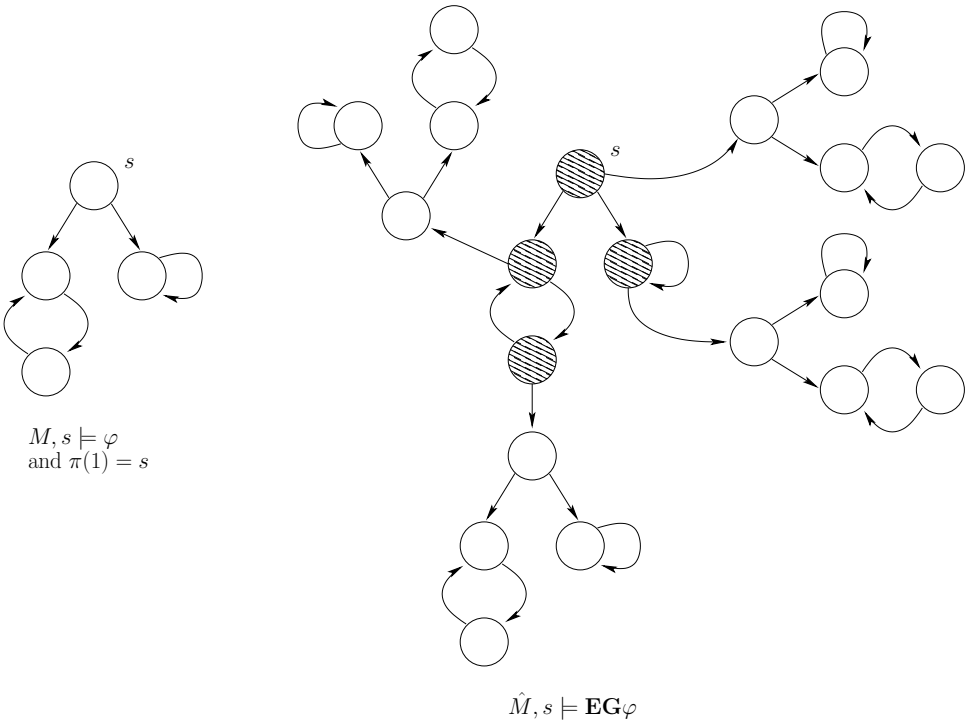


Figure 3.5: Example for the case **EG** in Theorem 11.

state $r \in S$ to the state s in each copy of M. Hence we get a global satisfiability in all states for φ. In Figure 3.5 we show an illustrating example.

- $M', s' \models \mathbf{E}(\varphi \mathbf{U} \psi)$ as we do not need to satisfy φ.
- $\tilde{M}, \tilde{s} \models \mathbf{A}(\varphi \mathbf{U} \psi)$ with the Kripke structure $\tilde{M'}$ constructed through M' in the same way as above.

If a CTL-operator containing **U** appears in a given formula $\varphi \in \mathrm{CTL}(B)$, then we need to check for the expression $\psi_1 \mathbf{U} \psi_2$ whether $\psi_2 = O_1 \dots O_k \lambda$. If $\lambda \in \{1, x, \neg x\}$ for an atomic proposition x, then $\psi_1 \mathbf{U} \psi_2$ is satisfiable, else not. To do this for a complete given formula, we need to start at the innermost nested **EU** and carry the satisfiability result over only pairwise for each **EU**-step.

Thus, for all CTL-operators, we only need to check for $\lambda \in \{0, 1, x, \neg x\}$ as in Theorem 10. If an **U**-operator appears, then we need to determine λ in the second formula-parameter. As we only need to search within the formula for specific expressions, which can be done in logarithmic space, we have proven the stated theorem for the clone N. The same holds for every clone $[B] \subseteq \mathrm{N}$, as we only need to check for unary functions. ■

To conclude our results from above, we showed a wider result than for CTL*-SAT with classifying the "unary Boolean functions" clone N. The other results for CTL*-SAT can be carried over to CTL-SAT. An illustration for these results can be found in Figure 3.6, where the red circles mark the PSPACE-complete sets and the green ones denote the sets solvable in L.

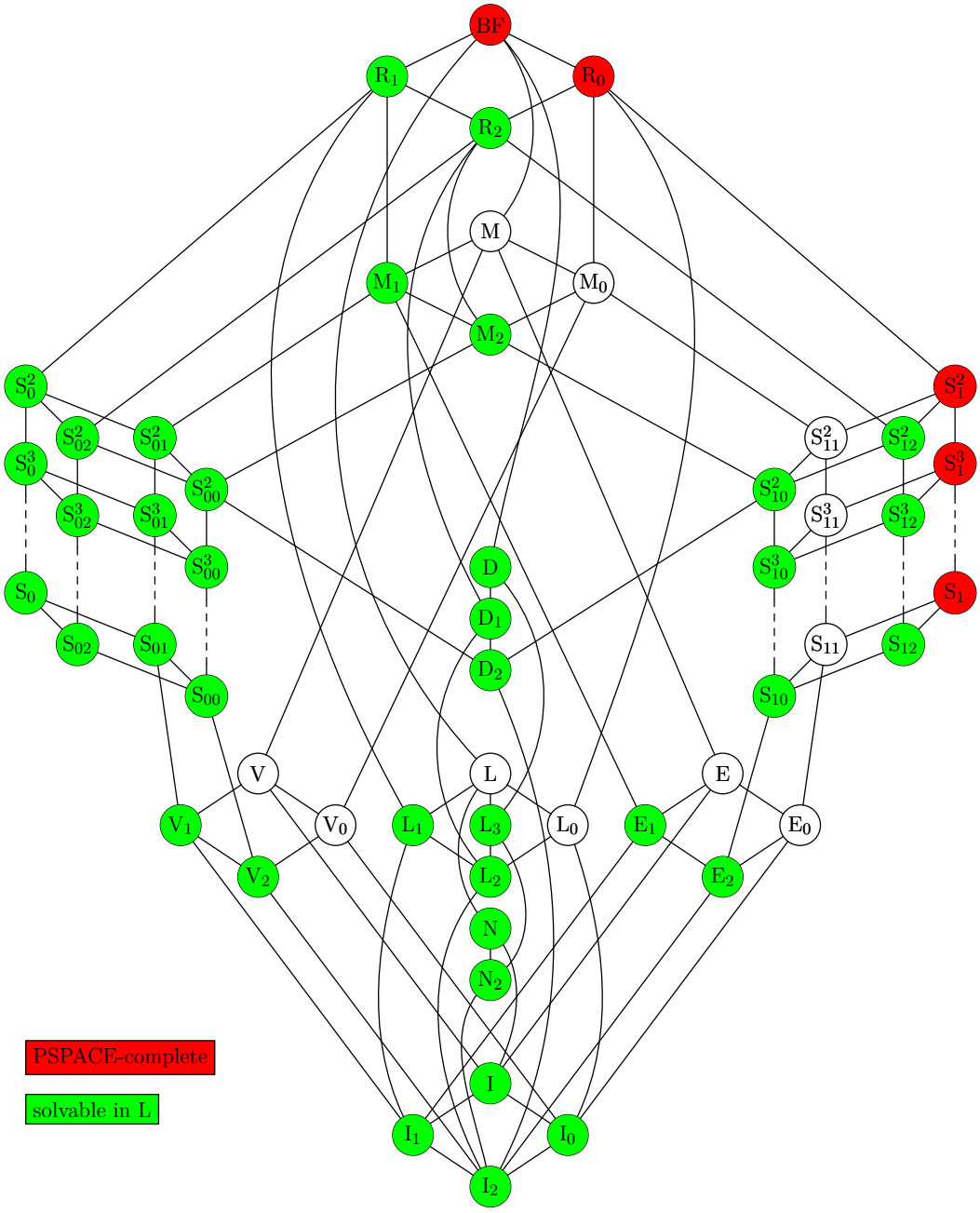


Figure 3.6: PSPACE-complete fragments for CTL-SAT($\{\mathbf{AX}, \mathbf{EX}\}$), and clones in L for CTL-SAT

3.3 Computation Tree Logic$^+$

In the end, we consider the branching time logic CTL$^+$ which in [JL03] is described as "allowing boolean combinations of path formulas inside a path quantifier but no nesting of them". In short words, the main difference to CTL* is, that we cannot use consecutively the temporal quantifiers without a path quantifier between them (e.g. $\mathbf{XX}\varphi$). Unlike for CTL, we can construct formulas as $\mathbf{E}(\mathbf{X}\varphi \vee \mathbf{X}\psi)$. Its complexity is of the same degree as CTL*, i.e., CTL$^+$ is 2-EXPTIME-complete.

In the following corollary, we will see which results of Section 3.1 can be transfered to this logic.

Corollary 11 *Let B be a set of Boolean functions.*

(1) CTL$^+$-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{BF})$ *is* PSPACE-hard.

(2) Let $\mathrm{S}_1 \subseteq [B]$. *Then* CTL$^+$-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B)$ *is* PSPACE-hard.

(3) Let $[B] \subseteq \mathrm{R}_1$. *Then* CTL$^+$-SAT$(B)$ *is in* L.

(4) Let $[B] \subseteq \mathrm{D}$. *Then* CTL$^+$-SAT(B) *is in* L. □

Proof (1) We can use the same reduction as in Theorem 1 because the obtained formula $\mathcal{F}$ is a CTL$^+$-formula.

(2) As proven for CTL*-SAT$(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, B)$ in Theorem 3, then we can use the same reduction since we just express the boolean functions of BF with the base of S_1. Hence the new formula will still be in CTL$^+$.

(3)+(4) Similar to the proof for Corollary 10. ■

Corollary 12 *Let B be a set of Boolean functions. Let* $\mathrm{S}_1 \subseteq [B]$. *Then* CTL$^+$-SAT$(B)$ *is* 2-EXPTIME-complete. □

Proof We use the fact of [JL03] that CTL$^+$-SAT is 2-EXPTIME-complete. As in the proof of Corollary 11 we can use the same reductions and pull the upper bound down to S_1. ■

3.4 Overview

In this section we will subsume the stated complexity results to get an outline about what is left. Altogether regarding CTL*-SAT and CTL$^+$-SAT, we could not state similar results to CTL-SAT with respect to the clone N and below. For the problems CTL$^\star(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\})$ and CTL$^+(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\})$, we leave the complexity-question open for the clones M and M_0, S_{11}^2 to S_{11}, V and V_0, L and L_0, E and E_0, N, and I. Regarding CTL-SAT$(\{\mathbf{AX}, \mathbf{EX}\})$, we have no result for M and M_0, S_{11}^2 to S_{11}, V and V_0, L and L_0, E and E_0.

function class (propositional operators)	complexity
below R_1 or below D	in L (always satisfiable)
above S_1	2-EXPTIME-complete
BF(all Boolean functions)	2-EXPTIME-complete

Table 3.1: Complexity results overview for CTL*-SAT

function class (propositional operators)	complexity
below R_1 or below D	in L (always satisfiable)
above S_1	PSPACE-hard
BF(all Boolean functions)	PSPACE-hard

Table 3.2: Complexity results overview for CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)

function class (propositional operators)	complexity
below N	in L
below R_1 or below D	in L (always satisfiable)
above S_1	EXPTIME-complete
BF(all Boolean functions)	EXPTIME-complete

Table 3.3: Complexity results overview for CTL-SAT

function class (propositional operators)	complexity
below N	in L
below R_1 or below D	in L (always satisfiable)
above S_1	PSPACE-complete
BF(all Boolean functions)	PSPACE-complete

Table 3.4: Complexity results overview for CTL-SAT($\{\mathbf{AX}, \mathbf{EX}\}$)

function class (propositional operators)	complexity
below R_1 or below D	in L (always satisfiable)
above S_1	2-EXPTIME-complete
BF(all Boolean functions)	2-EXPTIME-complete

Table 3.5: Complexity results overview for CTL$^+$-SAT

function class (propositional operators)	complexity
below R_1 or below D	in L (always satisfiable)
above S_1	PSPACE-hard
BF(all Boolean functions)	PSPACE-hard

Table 3.6: Complexity results overview for CTL$^+$-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)

	BF or $> S_1$	$< R_1$ or $< D$	$< N$
CTL*-SAT	2-EXPTIME-complete	in L	?
CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)	PSPACE-hard	in L	?
CTL-SAT	EXPTIME-complete	in L	in L
CTL-SAT($\{\mathbf{AX}, \mathbf{EX}\}$)	PSPACE-complete	in L	in L
CTL$^+$-SAT	2-EXPTIME-complete	in L	?
CTL$^+$-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$)	PSPACE-hard	in L	?

Table 3.7: Complexity results Overview

4 Conclusion and further work

Motivated by the straight $\leq_m^{\mathrm{p}}$-reduction from QBF to the Model-Checking problem in CTL*, we first wanted to state a $\leq_m^{\mathrm{p}}$-reduction from QBF to the Satisfiability problem for a fragment of CTL*, which we did in Theorem 1, and started this reduction at QBF-3SAT.

Hereupon we set our main goal of this thesis to achieve a mostly complete complexity classification for the temporal logics CTL, CTL*, and CTL$^+$, with respect to the structure of Post's Lattice and the Satisfiability problem (whether there exists a Kripke structure and a state, which fulfill a given temporal formula).

As written above, we got a PSPACE-hard fragment for the investigated logics, where we initially made restrictions to the only temporal operator ne**X**t, and were able to state a straight reduction from the well-known problem of quantified Boolean formulas (restricted to 3CNF) to the Satisfiability problem for the logic CTL$^\star(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\})$. This hardness result for PSPACE was the origin of our further considerations. Afterwards, we started to make restrictions to the used Boolean functions, and analyzed several clones in the lattice. We built up a chain of reductions from result to result, and finally reached the lower bound for PSPACE-hardness at S_1, as we show in Theorem 3. We also showed that decreasing the number of atomic propositions down to one single proposition in a CTL$^\star(\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}, \mathrm{BF})$-formula still keeps PSPACE-hardness. The step over to none atomic propositions leads to containment in P, see Theorem 5.

Furthermore, we could show that the reduction is valid for the logics CTL$^+$ and CTL, as well. Hence we can carry over the made hardness-results.

Then we stated in Theorem 6 the $\leq_m^{\mathrm{p}}$-reduction from M_0 to S_{11} for closing the gap to PSPACE-hardness as much as possible.

The next strong result is made for a huge set of clones (i.e., below R_1 and below D): they are all solvable by an algorithm that is only bounded by logarithmic space, which is a very satisfying result. Moreover for the restricted CTL-logic, we show that for the clones N and I (which are not contained in R_1 or D), both restricted logics are solvable in logarithmic space, too (see Theorem 10). Actually, we could show this complexity for the non-restricted problem CTL-SAT, too (cp. Theorem 11). Finally, we composed an algorithm which solves the problem CTL-SAT$(\{\mathbf{AX}, \mathbf{EX}\})$ in PSPACE, and hence stated a completeness result for that logic fragment in Theorem 9.

The most interesting fact we discovered, is the analogy between modal logic and the temporal logics restricted to the operator **X**. Many results in the modal world have a similar proof for this special temporal case – just few things must be adjusted (i.e., the cases for temporal operators). This analogy helps understanding the complexity structure of temporal logics from a different point of view. The reason for the connection between these two logics is the correspondence between Kripke structures in temporal world and frames in modal world, as well as the correspondence between the operators $\Diamond/\Box$ and **AX**/**EX**. Because of that, sometimes it is not very easy to state results for

the logics CTL* and CTL$^+$, as the path quantifiers are not necessary connected to the temporal operator ne**X**t. Especially for the possible completeness case: for CTL (where we have the operators **AX** and **EX**), we adjusted the known algorithm for the modal case, but were not able to state a similar result for the "non-operator-connected" logics. Here it is hard to check for the existence of a fulfilling Kripke structure, because we could fork in the structure with many adjoined **A**s and must not affect only the next states. Hence we cannot use the Algorithm 3.1 for these logics.

4.1 What further research should be done?

Consider the parts left open, we need to classify the problems CTL*-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$) and CTL$^+$-SAT($\{\mathbf{A}, \mathbf{E}, \mathbf{X}\}$) restricted to the following clones:

- M, M_0 (the clones with the monotonic functions),
- S_{11}^2 to S_{11} (the clones with the 1-seperating functions),
- V, V_0 (the clones with the logical *or*),
- L, L_0 (the clones with the logical *exclusive-or*),
- E, E_0 (the clones with the logical *and*),
- N, and I (the clone with the *negation*, and the clone with *identity*).

The first two items are maybe the next logical step in finding an evidence for PSPACE-hardness down to S_{11}. To achieve such a result, a reduction to the clone M must be found, which can be easily reduced to M_0 and with our Theorem 6 to S_{11} for keeping the same complexity degree.

We assume the complexity of the last four items is in the best case similar to its "surrounding" clones, also decidable in logarithmic space. At least there should be containment in P. We take the results for similar modal problems (cp. [BHSS06, pp. 6]) and our results for CTL-SAT as a strong evidence, but leave these problems open for further research.

Another very interesting step leads to the questions about a PSPACE-algorithm for the PSPACE-hard fragments of CTL*-SAT and CTL$^+$-SAT to achieve a completeness result, as we did for CTL-SAT. With respect to CTL-SAT, we need to classify the same clones as for CTL*-SAT except N and I.

Finally, similar research can be done for the Model-Checking Problem and the investigated logics, as well.

The main goal of classifying complexity degrees should be in getting a better understanding about the connections to other areas in theoretical computer science and mathematics. These connections help to apply new techniques from other areas to solve open problems from a new point of view, and that is a strong tool.

Bibliography

[BCRV03] Böhler, E., N. Creignou, S. Reith and H. Vollmer: *Playing with Boolean Blocks, Part I: Post's Lattice with Applications to Complexity Theory.* SIGACT News, 34(4):38–52, 2003.

[BdRV01] Blackburn, P., M. de Rijke and Y. Venema: *Modal logic.* Cambridge University Press, 2001.

[BHSS06] Bauland, M., E. Hemaspaandra, H. Schnoor and I. Schnoor: *Generalized Modal Satisfiability.* 23rd Annual Symposium on Theoretical Aspects of Computer Science, pages 500–511, February 2006.

[CE81] Clarke, E. M. and E. A. 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.

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

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

[Coo71] Cook, S. C.: *The complexity of theorem-proving procedures.* Third ACM Symposium on Theory of Computing, ACM, New York, pages 151–158, 1971.

[Dav04] Davis, M.: *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions.* Dover Publications, Incorporated, 2004.

[EJ00] Emerson, E. A. and C. S. Jutla: *The complexity of tree automata and logics of programs.* SIAM Journal on Computing, 29(1):132–158, February 2000.

[Eme90] Emerson, E. A.: *Temporal and modal logic.* Handbook of theoretical computer science (vol. B): formal models and semantics, pages 995–1072, 1990.

[End01] Enderton, H. B.: *A Mathematical Introduction to Logic.* Hartcourt/Academic Press, second edition edition, 2001.

[FL79] Fischer, M.J. and R.E. Ladner: *Propositional dynamic logic of regular programs.* Journal of Computer and Systems Sciences, 18:194–211, 1979.

[GMH81] Gannon, J., P. McMullin and R. Hamlet: *Data Abstraction Implementation, Specification, and Testing.* IEEE Trans. on Programming Languages and Systems, 3(3):211–223, July 1981.

[Hal95] Halpern, J. Y.: *The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic.* Artificial Intelligence, 75(2):361–372, 1995.

[Hem01] Hemaspaandra, E.: *The Complexity of Poor Man's Logic.* Journal of Logic and Computation, 11(4):609–622, 2001. Corrected version: [Hem05].

[Hem05] Hemaspaandra, E.: *The Complexity of Poor Man's Logic.* CoRR, cs.LO/9911014, 1999, 2005. Revised 2005, Correct version for: [Hem01].

[Hoa69] Hoare, C. A. R.: *An axiomatic basis for computer programming.* Communc. ACM, 12(10):576–580, 1969.

[Hol91] Holzmann, G. J.: *Design and validation of computer protocols.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[HP96] Holzmann, G. J. and D. Peled: *The State of SPIN.* In *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, pages 385–389, London, UK, 1996. Springer-Verlag.

[JL03] Johannsen, J. and M. Lange: *CTL$^+$ is Complete for Double Exponential Time.* In Baeten, J. C. M., J. K. Lenstra, J. Parrow and G. J. Woeginger (editors): *Proc. 30th Int. Coll. on Automata, Logics and Programming, ICALP'03*, volume 2719 of *Lecture Notes in Computer Science*, pages 767–775, Eindhoven, NL, 2003. Springer-Verlag.

[Lad77] Ladner, R. E.: *The Computational Complexity of Provability in Systems of Modal Propositional Logic.* SIAM Journal on Computing, 6(3):467–480, 1977.

[Lew79] Lewis, H.: *Satisfiability problems for propositional calculi.* Mathematical Systems Theory, 13:45–53, 1979.

[Pip97] Pippenger, Nicholas: *Theories of Computability.* Cambridge University Press, May 1997.

[Pnu77] Pnueli, A.: *The temporal logic of programs.* IEEE Symposium on Foundations of Computer Science (FOCS), pages 46–57, 1977.

[Pos41] Post, E.: *The two-valued iterative systems of mathematical logic.* Annals of Mathematical Studies, 5:1–122, 1941.

[Pra80] Pratt, V. R.: *A near-optimal method for reasoning about action.* Journal of Computer and System Sciences, 20(2):231–254, 1980.

[QS82] Queille, J.-P. and J. Sifakis: *Specification and verification of concurrent systems in CESAR.* In *Proceedings 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1982.

[Rei] Reith, S.: *Steffen Reith.* Website. Available online at `http://www.informatik.fh-wiesbaden.de/~reith/`.

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

[Sch01] Schöning, U.: *Theretische Informatik – kurzgefasst.* Spektrum Akademischer Verlag, 4. Auflage edition, 2001.

[Sch02] Schneider, T.: *Komplexität modaler Logiken.* Diplomarbeit, Friedrich-Schiller-Universität Jena, September 2002.

[Sip07] Sipser, M.: *Introduction to the Theory of Computation.* International Thomson Publishing, 2 edition, 2007.

[Sto77] Stockmeyer, L.: *The polynomial-time hierarchy.* Theoretical Computer Science, 3:1–22, 1977.

[VS85] Vardi, M. Y. and L. Stockmeyer: *Improved upper and lower bounds for modal logics of programs.* In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 240—251, New York, NY, USA, 1985. ACM Press.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.