Hybrid Branching-Time Logics

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

Vorgelegt im
Fachbereich Elektrotechnik/Informatik
der Universität Kassel

von
Daniel Kernberger

Kassel, Juli 2019

Erstgutachter:	Prof. Dr. Martin Lange
Zweitgutachter:	Prof. Dr. Thomas Schneider
Tag der Disputation:	31.10.2019

Abstract

In formal verification we are interested to prove that a given system works correctly with respect to some underlying property. This is usually done by checking if a mathematical model of this system satisfies a logical formula that specifies this property.

Modal and temporal logics – and especially branching-time logics ranging from CTL over CTL* up to the modal μ-calculus – have become a well-known logical formalism for this task over the last few decades. Formulas in branching-time logics like CTL and CTL* are easy to understand and to use and have a well-developed model theory with connections to tree automata. The latter enables conceptually simple decision procedures. The modal μ-calculus is more involved and not as easy to understand but makes up for this with its immense expressive power which even subsumes CTL* and many other branching-time logics and its decision procedures with astonishingly low complexities.

One reason for their well-developed theory is that all these logics are invariant under bisimulation. This property has enabled many decision procedures through the use of tree automata for satisfiability checking and model checking. However, it also limits the expressive power of all these branching-time logics in the same way. For example cycle detection, which is useful to test if a system can possibly get stuck in an infinite loop, or other similar properties cannot be tested with these logics.

This has led to numerous extensions of branching-time logics over the years that enriched these logics with specifically suited operators for the task at hand. One specific framework to extend such logics – which over the last one or two decades has been used to extend numerous logics, not only modal and temporal ones – are *hybrid logics*. This framework borrows ideas from first-order logic to enable more precise "structural" reasoning.

The idea for this framework is that some states in a system might be very important and should be easily identifiable. For example there might be some point in the system that absolutely needs to be executed in each computation or there is some point in the system that enables a deadlock and that should therefore never be reached etc. To highlight or identify such states, hybrid logics add the possibility to *name* these states and also identify them. Thus, essentially constants and variables similar to the same concepts in first-order logic are added together with a (restricted) way of manipulating them.

Early ideas for such a hybrid framework originated with Prior and Bull in the 1950's and 1970's. But the evolution of the hybrid framework used in this thesis dates back to the late 1990's and early 2000's to a series of papers by Goranko and has since been used to extend many modal and temporal but also some other related logics. The hybrid framework has proven to be quite powerful in its expressive power when added to a logic.

We add this hybrid framework systematically to the branching-time logics CTL, CTL^+, FCTL^+, CTL^* and the modal μ-calculus, study the decision procedures of these new *hybrid branching-time logics* and develop their model theory based on the already well-established model theory of the underlying branching-time logics. We start in Chapter 3 by defining these new hybrid logics and giving them a proper semantics. We also discuss various challenges that arise when adding the hybrid framework to branching-time logics.

Chapter 4 and 5 then deal with the model theory. We introduce a suitable notion of bisimulation that provides an upper bound for the expressive power of (most) hybrid logics presented in this thesis and develop several tools and techniques like Ehrenfeucht-Fraïssé Games that help us to establish and complete a picture of the relationships between these logics in terms of their expressive power.

Chapter 6 then deals with the model checking problem, i.e. the problem of checking if a mathematical model of a system given as a Kripke structure satisfies a property specified in one of these hybrid logics. We present a comprehensive analysis of this problem for all introduced hybrid logics with model checking algorithms and matching lower bounds showing that our algorithms are optimal. We also identify some fragments with lower model checking complexities.

The last chapter then deals with the satisfiability problem. Generally speaking, the hybrid framework mostly leads to undecidable satisfiability problems. We present an alternative undecidability proof that even works for the most basic hybrid logic considered in this thesis – hybrid CTL, even if the Next-operator is left our. We then continue to identify some decidable fragments.

Publications

Parts of this thesis have already been published in peer reviewed conference proceedings or journals. The publications – listed in chronological order – are:

[53] D. Kernberger and M. Lange. Model checking for the full hybrid computation tree logic. In *Proc. 23rd Int. Symp. on Temporal Representation and Reasoning, TIME'16*, pages 31–40. IEEE Computer Society, 2016.

[54] D. Kernberger and M. Lange. The fully hybrid mu-calculus. In *Proc. 24th Int. Symp. on Temporal Representation and Reasoning, TIME'17*, volume 90 of *LIPIcs*, pages 17:1–17:16. Dagstuhl-Leibniz-Zentrum, 2017.

[55] D. Kernberger and M. Lange. Model checking for hybrid branching-time logics. *Journal of Logical and Algebraic Methods in Programming*, 2018.

[56] D. Kernberger and M. Lange. On the expressive power of hybrid branching-time logics. In *25th International Symposium on Temporal Representation and Reasoning, TIME 2018*, pages 16:1–16:18, 2018.

Also, the following journal version which is due to appear later this year is also worth mentioning:

[57] D. Kernberger and M. Lange. On the expressive power of hybrid branching-time logics. *Theoretical Computer Science*, 2019. Submitted for publication.

The following account gives a detailed description which parts of this thesis have already been published in one of the above mentioned papers. Additionally the main contributions of the co-authors are mentioned.
First of all, all publications cover parts of Chapter 3 which introduces the hybrid logics that are the main topic of this thesis.

[53]/[55] contain the first results on model checking that can also be found in Sections 6.1.1-6.1.3 and 6.2.1-6.2.3. However, [55] massively improves upon these first results by also studying other logics and a more detailed analysis. The results presented in Sections 6.1.1-6.1.3 have been published in [55]. Sections 6.2.1, 6.2.2 and 6.2.3 are also covered in [55] but have been expanded with more details on the involved constructions.

[54] covers the logic presented and discussed in Section 3.5 as well as the results from Sections 4.2 and 4.3. Also, as a first application of the results, Theorem 5.35 was first presented in this publication.

[56]/[57] contain in most parts the same results. However [57] corrects some minor mistakes first made in [56] and features some more in-depth proofs. Both cover mostly Section 5.2 as well as the definition of the EF-games in Section 5.1 and Theorem 5.5. Theorem 5.16 has been slightly rewritten to cover some of the corrections made in [57].

Acknowledgements

First, I would like to thank my supervisor Martin Lange. For the last five years his guidance and support helped me immensely. He not only gave me the freedom and time to choose the right topic but also helped a lot in shaping this thesis. Out of every discussion with him arose new ideas or alternative perspectives that helped me in solving many problems and – often more importantly – countless new questions so that the possibilities for further research never ceased.

Secondly, I want to thank Prof. Dr. Thomas Schneider for agreeing to referee this thesis.

I would also like to thank my current and former colleagues: Milka Hutagalung, Lara Yörük and in particular Norbert Hundeshagen and Florian Bruse. Their doors were always open wide for any questions I had – either research or work or otherwise related. And – surprisingly – sometimes hours of discussions that start at the tiniest detail and lead to completely unrelated topics can simply help to take the mind off of seemingly unsolvable problems. I also want to thank Tina Landefeld and Michael Möller for all their help during my time here at the university.

Further thanks goes to my parents. Without them and their support throughout the years none of this would have been possible.

And finally, I want to thank my wife Kirsten who took every step of the journey with me and supported me in every decision I had to make along the way. She is without a doubt the foundation without which writing this thesis would not have been possible. Or in other words: "Frodo wouldn't have got far without Sam."

Declaration

I herewith give assurance that I completed this dissertation independently without prohibited assistance of third parties or aids other than those identified in this dissertation. All passages that are drawn from published or unpublished writings, either word-for-word or in paraphrase, have been clearly identified as such. Third parties were not involved in the drafting of the content of this dissertation; most specifically I did not employ the assistance of a dissertation advisor. No part of this thesis has been used in another doctoral or tenure process.

(Daniel Kernberger)

Contents

1 Introduction **1**

2 Preliminaries **9**

2.1 Kripke structures 9
2.2 Branching-Time Logics 11
2.3 Linear Temporal Logic 15
2.4 The Modal μ-calculus 16
2.5 Syntactic conventions 23
2.6 Expressiveness & A Branching-Time Hierarchy 25
2.7 Model Checking 26
2.8 Automata 28
2.9 Games 30
2.10 Computational Complexity 37
2.11 Tiling Problems 43

3 Hybridisation of Branching-Time Logics **47**

3.1 The Main Concepts of Hybrid Logic 48
3.2 Hybridisation of CTL* 51
3.3 A Syntactical Hierarchy and a Unifying Semantics 54
3.4 Hybridisation of CTL, CTL$^+$ and FCTL$^+$ 56
3.5 The Fully Hybrid μ-calculus 58
3.6 The Hybrid Branching-Time Landscape 65
3.7 Hybrid and other extensions of Branching-Time Logics 67

4 The Expressive Power of the Fully Hybrid μ-calculus **77**

4.1 The Polyadic μ-calculus and H_μ 78
4.2 Hybrid Bisimulation 82
4.3 H_μ and Hybrid Bisimulation 86
4.4 A hierarchy for bounded fragments of H_μ 89

5 The Hybrid Branching-Time Hierarchy **93**
5.1 Ehrenfeucht-Fraïssé Games 94
5.2 The Hierarchy on Kripke Structures 106
5.3 The Hierarchy on Tree Structures 124

6 Model-Checking for Hybrid Logics **131**
6.1 Lower bounds 131
6.2 Upper Bounds 147
6.3 Bounded Variable Fragments 184
6.4 A Complete Picture 188

7 Satisfiability and Decidable Fragments **191**
7.1 Undecidability of the Full Hybrid Logic 192
7.2 Decidable Fragments 195

8 Conclusion **201**

Bibliography **205**

Chapter 1

Introduction

Formal Verification and Temporal Logics. Mobile phones, tablets and computers in general have become important parts of our everyday life. We rely on our smartphones and its various applications for communication both in business and in private, for navigation via GPS, for financial services of our bank or even for basic time management such as using the alarm clock and many more things. Most of these applications are multi-layered and themselves rely on other functioning systems or background applications. Navigation apps, for example, calculate the optimal route to a destination but rely for the current location on GPS satellites and possibly also other services that for example provide real-time traffic data.
This means that we are more and more reliant on various electronic systems. Some of them are only for entertainment, some are important for our everyday life and some of them may even be safety-critical which means that malfunctioning of these systems may lead to life-threatening injuries or catastrophic environmental harm. A few examples of the latter category are aeroplane control systems, railway signalling, life-support systems in medicine or even control systems for nuclear reactors.
Thus it is *vital* that such systems are "correct" by design. However, for systems of such complexity it is not always clear what constitutes correct behaviour. In the navigation app for example it is quite clear what it means that the current location should be correct, but *the* optimal route to a destination is ambiguous. Does optimal mean shortest, fastest, nicest or most eco-friendly? The answer may vary depending on many factors.
Mathematical logics provide a framework to make unambiguous and precise statements – usually called logical formulas – and thus can often be used to precisely formulate what it means for a system to function "correctly". They are often used in the research area of *formal verification* which focusses on proving – beyond any doubt – that a system behaves "correctly" with the help

of formal methods like logical formulas. Another side of formal verification and mathematical logic is that these logical formulas are typically evaluated on an abstraction or a model of the real physical system. This means that formal verification can be especially helpful in tracing bugs and errors during the design process of such a real system.
Over the years a wide variety of logics have evolved to express different kinds of properties on various types of systems, often highly specialised to express very specific properties on special types of structures. An important and in the modern setting especially useful family among these logics are *temporal logics* which focus on making logical statements about the ongoing behaviour of nonterminating and interactive systems.
It is quite self-explanatory why it is especially important nowadays to express properties about interactive systems. All kinds of applications are governed by user input or sensor readings etc. and thus these systems need to be able to change their behaviour based on that input. It is however not self-evident at first why nonterminating systems are so important: at first, one might think that systems nowadays are replaced quite frequently by a new version or a whole new system alltogether. Thus, why would it be useful to design a system as if it would be running forever? While this may be true, usually one does not know in advance how long exactly the system needs to function correctly and thus especially when designing such a system it needs to be designed *as if it would run forever.*
Temporal logics can roughly be divided into two subgroups based on the underlying nature of time. First, there are *linear* temporal logics. These model time as if each time point has a *unique possible future.* Consequently, the models on which these logics are evaluated are linear. Typically this is seen as modelling a single computation of a system and, hence, linear temporal logics describe the behaviour of a single computation. A typical task for these kinds of logics is for example to test whether a specific computation is safe which means properties like "at no point in time it is the case that the system enters a critical state" are typical examples. For *branching* temporal logics each moment in time may have several possible futures, e.g. depending upon user input or other interactive controls, the computation might continue in different forms but of course a system designer needs to consider all possible behaviours to design a "correct" system. A typical statement might be "On all possible computations every user request is eventually answered".
In this work we focus mainly on branching-time logics. Aside from their branching aspect time is usually seen as moving forward in each step of the computation and thus typical models for these logics are infinite tree structures that describe this passing of time in each step as well as the branching model of time. However, for practical purposes this involves another step

of abstraction: not only from the real system to the abstract model of the system which may have cycles etc., but also from the abstract model to the set of computations of this model.

Most branching-time logics – especially the ones considered in this thesis – are designed or have evolved in such a way that this second step of abstraction is unnecessary: they simply cannot distinguish between the set of computations or the model itself and consequently can also be evaluated with respect to these models directly. In the design and verification process this has obvious advantages since it eliminates possible errors in the second abstraction step and makes these logics easier to use in general. And also from a theoretical point of view this property – called bisimulation-invariance – has many advantages. It opens up connections to many other research areas such as automata theory or game theory and in doing so enables the use of many decision procedures that are also of practical relevance.

But on the other hand this inherently limits the properties that can be expressed with these logics. For example, cyclic behaviour or the recurrence of a single potentially important state of the system can of course not be detected if the logic cannot distinguish between a cyclic model and its tree-shaped set of computations.

For this reason several extensions to those branching-time logics have been proposed that on the one hand try to retain the good properties of branching-time logics but on the other hand also add to the expressive power. One idea to overcome specifically the limitations mentioned above is to explicitly give *unique names* to the states of a system and a way to identify the name of a state. Early ideas of *naming* states have already been proposed in the 50's and 70's by Prior [78] and Bull [19] for precursors of modern branching-time logics.

Their concept was only picked up again in the late 90's by Goranko [40, 41, 42, 43] and from there evolved into its current iteration, which features not only the possibility to have *fixed* names for states but also to dynamically name states in the context of a logical formula. Moreover, one can *test* for any name and also refer to these named states. Various extensions of logics that feature these concepts and operators have been studied since then and are summarised under the name *hybrid logics*.

The above mentioned limitations like detecting cyclic behaviour are easily overcome with these principles: We can simply name a specific point in a computation and then test if it will be seen again later on. If this is the case then of course we have found a cycle. It turns out that this hybrid framework is quite powerful but at the same time conceptually nice and easy to use.

The term "hybrid" is used because these concepts are also reminiscent of similar concepts from first-order logic. Names for states of a system can also be seen as variables or constants in this logic and a limited way to work with them. Hence, extensions of logics with these hybrid concepts are also seen as crossover logics that mix features from two worlds.

Outline of this Thesis. In this thesis we discuss a way of extending classical branching-time logics with hybrid features to overcome the limitations in terms of their expressive power discussed above and study the effects and trade-offs in adding this additional expressive power.

The goal of this thesis specifically is to extend the framework of hybrid logics which features dynamic naming and referencing of states to the well-known branching-time logics CTL, CTL*, the modal μ-calculus and some of their fragments and then study the resulting hybrid branching-time logics. We are particularly interested in how the hybridisation of these logics affects their expressive power, the model checking problem and also the satisfiability problem of these logics. The thesis is structured in the following way.

Chapter 2 introduces various basic concepts, logics and general ideas that are used throughout the thesis. In particular, we present the branching-time logics from CTL up to CTL* as well as the modal μ-calculus, together with a short section about the theory of least and greatest fixed points which will later help to understand the hybridisation of the modal μ-calculus. These logics form the basis of the hybrid logics studied in later chapters. Additionally, we provide the necessary definitions and background for related research areas such as Büchi automata, 2-player games and computational complexity as it is needed for the understanding of the later parts of this thesis.

Chapter 3 then introduces the hybrid branching-time logics which are the main topic of this thesis. The hybrid operators are discussed and added to the previously defined branching-time logics. We discuss several possibilities, challenges and effects when adding these operators to the standard branching-time logics. In the context of branching-time logics like CTL* we obtain several possibilities of how these hybrid operators can be added and in the context of least and greatest fixed points we discuss challenges in defining a proper semantics when adding hybrid operators to the μ-calculus. The chapter finishes with an overview of the obtained logics as well as a short comparison to other recent extensions of branching-time logics to obtain a first impression of the capabilities of hybrid logics.

In Chapter 4 and 5 we deal with the expressive power of the obtained hybrid logics. First, Chapter 4 mainly focusses on the hybrid extension of the modal μ-calculus. The Hybrid μ-calculus is compared to another well-known extension of the modal μ-calculus– the *polyadic* μ-calculus. Then we introduce a refined bisimulation relation – called k-bisimulation – that is specifically tailored towards hybrid logics and show that this notion in a sense captures the limit of the expressive power of the hybrid μ-calculus in that no formula (which uses at most k names) can distinguish structures that are k-bisimilar. A first application of this then shows that allowing more variables truly increases the expressive power of such hybrid logics.

Chapter 5 then compares all hybrid branching-time logics introduced in this thesis in terms of their relative expressive power, i.e. which properties can be expressed by any of these hybrid logics and which cannot be expressed. We develop and extend several techniques to show the precise relationships between these logics. This includes several translations between these hybrid logics as well as a new kind of Ehrenfeucht-Fraïssé game that helps to distinguish which properties cannot be expressed by some hybrid logics. In doing so, we obtain a hierarchy of expressive power. The chapter concludes with an overview over the obtained relationships and a short excursion to the semantically restricted logics on trees for which the hierarchy changes quite a bit.

Chapter 6 is devoted to the model checking problem of all the hybrid logics. We analyse each logic and provide lower and upper bounds for the complexity of their respective model checking problem. The lower bounds are mostly obtained by encoding suitable instances of tiling problems, which, once again, prove to be a very useful tool. For the upper bounds we utilise and combine techniques from various adjacent fields: Simple extensions of non-hybrid algorithms, game-based approaches, automata constructions and reductions to the non-hybrid case. Since the complexity of the model checking problem rises in most cases compared to the non-hybrid logic, we also take a look at fragments with lower complexity: bounded fragments. These are the logics in which the number of names for states is restricted. We also show that these fragments are computationally as hard as the non-hybrid versions. Once again, the chapter concludes with an overview over the obtained results.

Finally, Chapter 7 touches on the satisfiability problem of hybrid logics. Generally, this problem is undecidable for hybrid logics – even for the most basic hybrid modal logic which is subsumed by any hybrid logic considered in this thesis. For this reason our research mainly focusses on fragments that are still decidable. We present some of these fragments and in doing so provide a rough overview over possible future directions for more research in this area. Since this hybrid framework is so powerful, the decidable fragments are quite limited in their expressive power. We use some results obtained in the thesis to slightly extend already known decidable fragments.

The thesis then concludes with a summary of the obtained results and some remarks about future research in the area of hybrid logics.

Related Works. As already mentioned above, first encounters with the idea of names for states date back to the 50's and 70's [78, 19] but the idea only gained traction in the late 90's with a series of papers by Goranko [40, 41, 42, 43] who specifically developed the current framework and added these concepts first to modal and then also to basic temporal logics.
This sparked a great interest in hybrid logics, predominantly hybridisations of modal logics and some baisc temporal extensions of these modal logics.
Shortly after the ideas for hybrid modal and temporal logics were also picked up in [14] and later in [5, 6, 7, 13, 6]. These papers predominantly studied the complexity of the satisfiability problem for hybrid modal and temporal logics and many variants and fragments but also developed a model theory for these logics. It was also first shown that adding dynamic naming of states to modal logics leads to undecidability even when done in quite restrictive ways. This result was further sharpened by Marx in [69] who added hybrid machinery to the description logic $\mathcal{ALC}$ – a close relative of modal logics – and showed that while the added expressive power might be very useful and desirable, even with only one available name the satisfiability problem for $\mathcal{ALC}$ with self reference becomes undecidable.
In parallel Sattler and Vardi picked up the idea of hybrid machinery and tried to add them to the modal μ-calculus [80]. However, they only add static names and references to these names and forego dynamic naming of states to obtain a still decidable logic. However, they show that even such an extended μ-calculus has still an ExpTime-complete satisfiability problem and thus is only as complex as the modal μ-calculus without this hybrid machinery.
The above mentioned undecidability results in the presence of dynamic naming immediately started the search for decidable fragments that still feature some form of dynamic names for states. In [89] a connection between hybrid formulas without any occurrences of universal modalities in the scope of dynamic naming in the scope of another universal modality and the $\forall$-guarded fragment of first-order logic [44, 88] which lead to a 2-ExpTime decision procedure for this fragment. Later [10] extended the ideas to show decidability of a similarly structured fragment of hybrid CTL. At the same time [38] studies the semantic restriction of hybrid logics on linear structures and regains decidability but with a nonelementary complexity for the full logic and also obtains various decidable fragments with complexities ranging from NP to PSpace depending on the types of temporal and hybrid operators that are allowed.
Both [89] and [38] also begin to study the model checking problem for these hybrid logics. Shortly after in [37] a detailed study of the model checking problem was published. In general hybrid logics with dynamic naming have

a PSPACE-complete model checking problem while the complexity remains in P if dynamic naming is forbidden. Lange [62] later improved upon their techniques and presented a local model checking procedure based on games. In [96] it was the first time since Goranko's paper [43] that the branching-time logic CTL with added hybrid machinery was extensively studied. It was proven that hybrid CTL interpreted only on tree structures has a 2-EXPTIME-complete satisfiability problem via a connection to alternating Büchi-automata. This connection to automata theory was further used in [83] to show that on linear structures the one variable fragment of hybrid temporal logic has an EXPSPACE-complete satisfiability problem and model checking can be done in P. Later in [52] the same authors also discuss the hybrid extension of CTL$^+$ and its relationship to hybrid CTL but again, only interpreted on tree structures.
Lastly, in [74, 70] the authors again look at the satisfiability problem of hybrid modal logic but this time over transitive frames and frames based on equivalence relations and different combinations of hybrid and temporal operators to better understand the boundaries of decidability for hybrid logic.

Chapter 2

Preliminaries

In this chapter we give some basic definitions and notation that will be used throughout the thesis. This chapter can be roughly partitioned into two parts.
The first half of this chapter introduces the logical background that is needed in this thesis. We introduce the relevant branching-time logics ranging from CTL to CTL* that are the foundations for most hybrid logics in this thesis and also give a comprehensive introduction to fixed points as logical operators and the modal μ-calculus.
The second part of this chapter then introduces various related topics and frameworks that will be used throughout this thesis. We begin with a framework for comparing the expressiveness of various logics followed by a short section about basic terminology for the model checking problem. We then continue to define Büchi automata and some related automaton models for ω-regular languages and also present a basic framework and terminology for 2-player games. The chapter concludes with a short section about computational complexity and tiling problems which are a convenient way to prove hardness results for certain complexity classes and are used in this thesis to prove lower bounds for some model checking problems.

2.1 Kripke structures

Let $\mathit{Prop} = \{p, q, \ldots\}$ be a countable set of atomic propositions.

Definition 2.1. A *Kripke structure* is a tuple $\mathcal{K} = \langle S, \to, L \rangle$ such that S is a set of states, $\to \subseteq S \times S$ is a total transition relation, i.e. for every state $s \in S$ there is another state $t \in S$ such that $s \to t$ and $L : \mathit{Prop} \to 2^S$ is a labeling function that assigns to each atomic proposition p a set of states on which this proposition holds true.

If not stated otherwise, we usually assume that for a fixed Kripke structure $L(p) \neq \emptyset$ for only finitely many p. By $\mathbb{K}$ we denote the class of all Kripke structures.

A *path* π in a Kripke structure $\mathcal{K}$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that $s_i \rightarrow s_{i+1}$ for every $i \in \mathbb{N}$. For a path π we write π^i to denote the i-th state on this path and we refer to the index i as the i-th moment on a path. Likewise, we denote with π^I for some Interval $I = [m, n]$ the finite sequence of states $\pi^m \ldots \pi^n$ and with π^I for some half open Interval $I = [m, \infty)$ the subpath $\pi^m, \pi^{m+1} \ldots$. Moreover, the set of all paths in a given Kripke structure $\mathcal{K}$ is denoted by $\mathsf{Paths}(\mathcal{K})$.

Definition 2.2. A Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ is called *finite* if S is a finite set. We denote the class of all finite Kripke structures by $\mathbb{F}$.
The size of a finite Kripke structure $|\mathcal{K}|$ is given by $|Q| + |\rightarrow|$.

Definition 2.3. A *tree* is a Kripke structure $\mathcal{T} = \langle T, \rightarrow, L\rangle$ with $T \subseteq \mathbb{N}^*$ such that if $w \cdot l \in T$ with $w \in \mathbb{N}^*$ and $l \in \mathbb{N}$, then $w \in T$. Further, if $w \cdot l \in T$, then $w \cdot l' \in T$ for all $0 \leq l' < l$.
ε is called the *root* of the tree and for all $l \in \mathbb{N}$ and $w \in T$, $w \cdot l \in T$ is called a *child* of w. A tree with $T \subseteq \{0\}^*$ is called a *word structure*.
We denote the class of all trees by $\mathbb{T}$ and the class of all word structures by $\mathbb{W}$.

Definition 2.4. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure. We define the *tree-unfolding* of $\mathcal{K}$ rooted at $s \in S$ as $\mathcal{T}_{\mathcal{K},s} := \langle S^*, \rightarrow, L^*\rangle$ with $S^* := \{\pi^{[0,m]} \mid m \in \mathbb{N}, \pi \in \mathsf{Paths}(\mathcal{K}), \pi^0 = s\}$ the set of all finite paths in $\mathcal{K}$, the transition relation $\rightarrow$ is induced by the immediate prefix relation on finite paths, i.e. $\pi^{[0,m]} \rightarrow \pi^{[0,m+1]}$ for all paths π and $m \in \mathbb{N}$ and the labeling function $L^*(p) := \{\pi^{[0,m]} \mid \pi^m \in L(p)\}$ simply uses the label of the last state on a finite path.

It is not hard to see that for Kripke structures with a finite branching degree, i.e. for those where the maximum number of successors of any state is finite, the tree-unfolding is also a tree with finite branching-degree.
One of the most important notions for Kripke structures in the realm of branching-time logics is *bisimulation*. A bisimulation relates states of two Kripke structures and – as the name suggests – it captures the idea that each computational step or behaviour in one of the structures can be simulated by the other structure and vice versa.

Definition 2.5. Let $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0\rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1\rangle$ be two Kripke structures. A *bisimulation relation* between $\mathcal{K}_0$ and $\mathcal{K}_1$ is a non-empty binary relation $\sim \subseteq S_0 \times S_1$ such that for all tuples $s_0 \sim s_1$ we have:

(prop) For all $p \in \mathit{Prop}$ we have $s_0 \in L_0(p)$ if and only if $s_1 \in L_1(p)$.

(zig) For every $s'_0 \in S_0$ such that $s_0 \rightarrow_0 s'_0$ there is a $s'_1 \in S_1$ with $s_1 \rightarrow_1 s'_1$ such that $s'_0 \sim s'_1$.

(zag) For every $s'_1 \in S_1$ such that $s_1 \rightarrow_1 s'_1$ there is a $s'_0 \in S_0$ with $s_0 \rightarrow_0 s'_0$ such that $s'_0 \sim s'_1$.

We say that $\mathcal{K}_0$ and $\mathcal{K}_1$ are *bisimilar* or *bisimulation equivalent* or short $\mathcal{K}_0 \sim \mathcal{K}_1$ if there exists a nonempty bisimulation relation between $\mathcal{K}_0$ and $\mathcal{K}_1$. Moreover, we say that two states $s_0 \in S_0$ and $s_1 \in S_1$ are *bisimilar* or *bisimulation equivalent* if there exists a bisimulation relation $\sim$ between $\mathcal{K}_0$ and $\mathcal{K}_1$ such that $s_0 \sim s_1$.

Example 2.6. Let $\mathcal{K} = \langle S, \rightarrow, L \rangle$ be a Kripke structure and let $\mathcal{T}_{\mathcal{K},s}$ be the tree-unfolding from some state $s \in S$. Then $\mathcal{K} \sim \mathcal{T}_{\mathcal{K},s}$.
The bisimulation relation $\sim$ relates a state $t \in S$ with a finite path $\pi^{[0,m]}$ if and only if $\pi^m = t$. The (prop) clause thus is satisfied easily since the labelling of $\pi^{[0,m]}$ is given by its last state which is t. And similarly the (zig) and (zag) conditions are satisfied as well by the construction of the transition relation of $\mathcal{T}_{\mathcal{K},s}$.

2.2 Branching-Time Logics

Temporal logics are usually extensions of modal logics that are equipped with operators to talk about the future behaviour of a system. Both the behaviour after a finite amount of time as well as properties about a possible infinite behaviour can typically be expressed. There are two broad categories of temporal logics: First, there are linear temporal logics like LTL that are usually used to model the behaviour of a *single* (possibly infinite) computation. Their model of time as the name suggests is linear, meaning that each possible moment in time has a *unique* possible future. And secondly, there are branching-time logics. For branching-time logics the underlying model of time features many possible futures. Thus, any moment in time may have various different possible futures. These logics are widely used to state properties about *all* or *some* computations of a given system.
As extensions of modal logics both categories strive to retain the good properties of modal logic like decidability, low computational complexity, finite- and tree-model properties etc. while also enhancing the quite limited expressive power of modal logics. We will mainly focus on the second category of temporal logics in this thesis, i.e. branching-time logics. The following section introduces the relevant branching-time logics.

CTL. We begin with the most basic branching-time logic: CTL. Formulas of CTL are generated by the grammar

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\varphi \mid \mathsf{AX}\varphi \mid \mathsf{A}\varphi\mathsf{U}\varphi,$$

where $p \in \mathit{Prop}$. Branching-time formulas are interpreted with respect to a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ and a state $s \in S$. The satisfaction relation $\models$ is given by mutual induction as follows:

$$\begin{array}{lll}
\mathcal{K}, s \models p & \text{iff} & s \in L(p), \\
\mathcal{K}, s \models \neg\varphi & \text{iff} & \mathcal{K}, s \not\models \varphi, \\
\mathcal{K}, s \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathcal{K}, s \models \varphi_1 \text{ or } \mathcal{K}, s \models \varphi_2, \\
\mathcal{K}, s \models \mathsf{EX}\varphi & \text{iff} & \text{there exists a state } t \text{ with } s \rightarrow t \text{ and } \mathcal{K}, t \models \varphi, \\
\mathcal{K}, s \models \mathsf{E}\varphi_1\mathsf{U}\varphi_2 & \text{iff} & \text{there exists a path } \pi \text{ in } \mathcal{K} \text{ with } \pi^0 = s \text{ and } j \in \mathbb{N} \\
 & & \text{such that } \mathcal{K}, \pi^k \models \varphi_1 \text{ for all } k < j \text{ and } \mathcal{K}, \pi^j \models \varphi_2, \\
\mathcal{K}, s \models \mathsf{AX}\varphi & \text{iff} & \text{on all states } t \text{ with } s \rightarrow t \text{ it holds that } \mathcal{K}, t \models \varphi, \\
\mathcal{K}, s \models \mathsf{A}\varphi_1\mathsf{U}\varphi_2 & \text{iff} & \text{on all paths } \pi \text{ in } \mathcal{K} \text{ with } \pi^0 = s \text{ there is } j \in \mathbb{N} \\
 & & \text{such that } \mathcal{K}, \pi^k \models \varphi_1 \text{ for all } k < j \text{ and } \mathcal{K}, \pi^j \models \varphi_2.
\end{array}$$

We will use standard abbreviations like $\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\mathtt{tt} := p \vee \neg p$, $\mathtt{ff} := p \wedge \neg p$. Further, we will use standard temporal operators like $\mathsf{EF}\varphi := \mathsf{E}\mathtt{tt}\mathsf{U}\varphi$, resp. $\mathsf{AF}\varphi := \mathsf{A}\mathtt{tt}\mathsf{U}\varphi$ – which characterises reachability on at least one, resp. all paths, or $\mathsf{EG}\varphi := \neg\mathsf{AF}\neg\varphi$, resp. $\mathsf{AG}\varphi := \neg\mathsf{EF}\neg\varphi$ – which means that φ holds everywhere on at least one path, resp. all paths.

Example 2.7. The formula $\mathsf{AGEF}p$ states that "on all paths" or "at any time for any given execution" there is the possibility to reach a state where the property p holds.

We will refer to the operators E, A as *path quantifiers* and to the operators $\mathsf{X}, \mathsf{U}, \mathsf{F}, \mathsf{G}$ as *temporal operators.*
The syntax of CTL is quite restricted. It requires exactly one temporal operator directly underneath each path quantifier. Relaxing this constraint will lead us to richer branching-time logics.
To help us in doing so, we rewrite the grammar that produces CTL formulas and split the use of path quantifiers and temporal operators:

$$\begin{array}{lll}
\varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi, \\
\psi & ::= & \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi.
\end{array}$$

Formulas of type φ are referred to as *state* formulas. Formulas of type ψ are called *path* formulas. Path formulas can only occur as genuine subformulas in branching-time formulas.
Extending the grammar for path formulas will lead us to more expressive branching-time logics.

CTL$^+$. The logic CTL$^+$ extends CTL by also allowing boolean operators as path formulas. Thus the grammar that produces CTL$^+$ formulas is the following:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi,\\ \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi.\end{aligned}$$

The satisfaction relation for CTL$^+$ formulas is now twofold: State-formulas are – as in CTL – interpreted with respect to a Kripke structure $\mathcal{K}$ and a state s.

$\mathcal{K}, s \models p$	iff	$s \in L(p)$,
$\mathcal{K}, s \models \neg\varphi$	iff	$\mathcal{K}, s \not\models \varphi$,
$\mathcal{K}, s \models \varphi_1 \vee \varphi_2$	iff	$\mathcal{K}, s \models \varphi_1$ or $\mathcal{K}, s \models \varphi_2$,
$\mathcal{K}, s \models \mathsf{E}\psi$	iff	there is a path π with $\pi^0 = s$ such that $\mathcal{K}, \pi, 0 \models \psi$,
$\mathcal{K}, s \models \mathsf{A}\psi$	iff	on all paths π with $\pi^0 = s$ it holds that $\mathcal{K}, \pi, 0 \models \psi$.

Path-formulas are interpreted with respect to a Kripke structure $\mathcal{K}$, a path π in $\mathcal{K}$ and a moment k on this path:

$\mathcal{K}, \pi, k \models \varphi$	iff	$\mathcal{K}, \pi^k \models \varphi$,
$\mathcal{K}, \pi, k \models \neg\psi$	iff	$\mathcal{K}, \pi^k \not\models \psi$,
$\mathcal{K}, \pi, k \models \psi_1 \vee \psi_2$	iff	$K, \pi, k \models \psi_1$ or $\mathcal{K}, \pi, k \models \psi_2$,
$\mathcal{K}, \pi, k \models \mathsf{X}\varphi$	iff	$\mathcal{K}, \pi, k+1 \models \varphi$,
$\mathcal{K}, \pi, k \models \varphi_1\mathsf{U}\varphi_2$	iff	there exists $j \in \mathbb{N}$ with $j \geq k$ such that $\mathcal{K}, \pi, j \models \varphi_2$ and for all $k \leq i < j$: $\mathcal{K}, \pi, i \models \varphi_1$.

Example 2.8. The CTL$^+$ formula $\mathsf{E}(\mathsf{F}p_1 \wedge \mathsf{F}p_2 \wedge \ldots \wedge \mathsf{F}p_n)$ states that there is a path on which there are reachable states that satisfy a proposition p_i, $1 \leq i \leq n$.

Despite having boolean connectives as part of the path formulas one can show that CTL$^+$ is not more expressive than CTL. We will show an extended version of this in Section 5.2 that even holds for the hybrid variants of CTL and

CTL$^+$. However, one can show that CTL$^+$ formulas can be more succinct. For example one can show that any CTL formula that expresses the same property as φ in the example above must be at least exponentially larger than φ, c.f. [98, 1, 61].

FCTL$^+$. To truly increase the expressive power we will allow nesting of path formulas. However, full nesting of path formulas is in many cases too much. Often "fairness" constraints that are able to express that something holds infinitely often along some computation are sufficient.
This fairness constraint can naturally be expressed with our temporal operators by nesting the temporal operators G and F: the formula $\mathsf{EGF}p$ intuitively states that along some path it always holds that after finitely many steps p is reached, i.e. p holds infinitely often along this path.
The logic Fair CTL$^+$ – or short FCTL$^+$ – is obtained by adding this "infinitely often" construct as a native operator to CTL$^+$. Thus, the grammar for FCTL$^+$ is the following:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi,\\ \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi \mid \mathsf{GF}\varphi.\end{aligned}$$

The satisfaction relation from CTL$^+$ gets extended to the new operator as intended:

$$\mathcal{K}, \pi, k \models \mathsf{GF}\varphi \text{ iff there exist infinitely many } j \in \mathbb{N} \text{ with } \mathcal{K}, \pi, j \models \varphi.$$

Fairness properties cannot be expressed by CTL or CTL$^+$ and thus FCTL$^+$ is more expressive than CTL$^+$ [30].

Example 2.9. The formula $\mathsf{A}(\mathsf{GFrequest} \to \mathsf{GFanswer})$ states that on all paths, if a $\mathsf{request}$ is made infinitely often, then there are also infinitely many $\mathsf{answers}$. Thus, this formula expresses a (weak) fairness constraint that every request is eventually answered.

CTL*. CTL* now allows full LTL-like nesting of temporal operators on path formulas and thus increases the expressive power even further. Its grammar is given by:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi\\ \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi,\end{aligned}$$

where $p \in \mathit{Prop}$.

State formulas are interpreted as before. The satisfaction relation for path formulas with respect to a Kripke structure $\mathcal{K}$, a path π and a moment k on this path is as follows:

$$
\begin{array}{lll}
\mathcal{K}, \pi, k \models \varphi & \text{iff} & \mathcal{K}, \pi^k \models \varphi, \\
\mathcal{K}, \pi, k \models \neg\psi & \text{iff} & \mathcal{K}, \pi^k \not\models \psi, \\
\mathcal{K}, \pi, k \models \psi_1 \vee \psi_2 & \text{iff} & K, \pi, k \models \psi_1 \text{ or } \mathcal{K}, \pi, k \models \psi_2, \\
\mathcal{K}, \pi, k \models \mathsf{X}\psi & \text{iff} & \mathcal{K}, \pi, k+1 \models \psi, \\
\mathcal{K}, \pi, k \models \psi_1 \mathsf{U} \psi_2 & \text{iff} & \text{there exists } j \in \mathbb{N} \text{ with } j \geq k \text{ such that} \\
 & & \mathcal{K}, \pi, j \models \psi_2 \text{ and for all } k \leq i < j\text{: } \mathcal{K}, \pi, i \models \psi_1.
\end{array}
$$

Example 2.10. The formula $\mathsf{AF}(p \wedge \mathsf{X}p)$ states that on all paths there is a moment such that the next two states both satisfy p. One can show that this formula cannot be expressed by FCTL$^+$ and thus CTL* is more expressive than FCTL$^+$ [30, 24].

We freely use the usual propositional abbreviations for $\mathtt{tt}$, $\mathtt{ff}$, $\wedge$ as well as the temporal ones $\mathsf{F}\psi := \mathtt{tt}\mathsf{U}\psi$, $\mathsf{G}\psi := \neg\mathsf{F}\neg\psi$, $\varphi\mathsf{R}\psi := \neg(\neg\varphi\mathsf{U}\neg\psi)$ etc.

2.3 Linear Temporal Logic

The second broad category of temporal logics deals with single computations and thus the logics in it are usually interpreted over traces of a computational system or more abstractly infinite words. This thesis does not focus on single computational traces but rather on the branching side of temporal logics. However, when dealing with logics like CTL* that have LTL more or less embedded into them as path formulas we sometimes need to deal with certain concepts used in the well-established theory around LTL as well.
We formally define the logic here, but refer for more details to [77, 24].
Formulas of LTL or linear temporal logic are generated by the grammar

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi.$$

We already see by its syntactic definition that LTL is subsumed by CTL* path formulas.
LTL formulas are usually interpreted relative to an infinite word or a trace of a computation. In our case, we will only use LTL along paths of a Kripke structure – which of course can also be seen as infinite words over their labels. Consequently LTL formulas simply inherit their interpretation from CTL* path formulas.

Example 2.11. The LTL formula $\mathsf{FG}p$ is satisfied along a path of a Kripke structure if and only if there is a moment on this path such that from this moment on every state along this path satisfies p.

Further we say that a state $s \in S$ in a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ satisfies an LTL formula φ, or short $\mathcal{K}, s \models \varphi$, if $\mathcal{K}, s \models \mathsf{E}\varphi$, i.e. a state s satisfies an LTL formula if *there is a path* starting at s that satisfies the formula.

Remark 2.12. It is common that the semantics of LTL formulas with respect to a state in a Kripke structure is given in such a way that a state satisfies an LTL formula if and only if *all paths* starting from this state satisfy the formula. However, LTL is closed under negation so this definition is equivalent to the one we use in this thesis. Moreover, we only use LTL in this thesis at specific points to handle path formulas of CTL*. Usually we consider CTL* formulas to be in a normal form that only allows one path quantifier – the existential one. To better match this normal form we have chosen this (slightly unusual) semantics to stay consistent in our terminology.

2.4 The Modal μ-calculus

The modal μ-calculus L_μ is a modal logic enhanced with least and greatest fixed points. Fixed point constructs add a lot of expressive power to the very basic modal language. For example it has been shown that CTL* can be fully embedded into the μ-calculus [23]. The μ-calculus can also express properties that CTL* cannot. For example reachability in an even number of steps is easily expressible in the μ-calculus but not in CTL*. Thus, for many considerations the μ-calculus has become *the* branching-time logic in terms of its expressive power.
Its appeal however is not only founded in its expressiveness. As an extension of modal logic it inherits many nice properties like bisimulation-invariance or the tree-model property. Its decision procedures retain a somewhat surprisingly low complexity. Satisfiability is only ExpTime-complete. This is surprising because satisfiability for CTL*, which is subsumed by the μ-calculus, is already 2-ExpTime complete. Also its model checking problem is known to be in NP $\cap$ Co-NP [48]. However most practically relevant model checking algorithms available today – which in most cases solve the associated model checking parity game which we introduce in Section 2.9.3 – still run in exponential time, c.f. [49, 94, 82, 51, 39].
However, the search for a possible model checking algorithm that runs in polynomial time on all inputs continues. Recent advances in the area achieved

algorithms with a worst-case runtime in quasi-polynomial time, c.f. [20, 50, 34, 68]. However, a recent evaluation of these algorithms [90] suggests that these new quasi-polynomial algorithms generally still perform worse than the classical algorithms.
In this section we will introduce the basic concepts and ideas for fixed points as logical operators and their use in the modal μ-calculus. The theory about fixed points will also provide a good foundation to understand the hybridisation of the μ-calculus in Section 3.5.

2.4.1 Modal Logic

We first define *Basic Modal Logic* with second-order variables. For this, let $\mathit{Var}_2 = \{X, Y, \ldots\}$ be a countable set of second-order variables. Formulas of basic modal logic are derived from the following grammar:

$$\varphi := p \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond\varphi$$

where $p \in \mathit{Prop}$ and $X \in \mathit{Var}_2$. The semantics of modal logics – in contrast to the semantics of branching-time logics – are usually given from a global perspective. The meaning of a modal logic formula is a set of states – the set of states in a Kripke structure that satisfy this formula. Formulas of this logic are thus interpreted with respect to a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ and an assignment $\rho : \mathit{Var}_2 \rightarrow 2^S$ for the free second-order variables:

$$\begin{aligned}
[\![p]\!]^{\mathcal{K}}_{\rho} &= \{s \in S \mid s \in L(p)\},\\
[\![X]\!]^{\mathcal{K}}_{\rho} &= \rho(X),\\
[\![\neg\varphi]\!]^{\mathcal{K}}_{\rho} &= \{s \in S \mid s \notin [\![\varphi]\!]^{\mathcal{K}}_{\rho}\},\\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}}_{\rho} &= [\![\varphi_1]\!]^{\mathcal{K}}_{\rho} \cup [\![\varphi_2]\!]^{\mathcal{K}}_{\rho},\\
[\![\Diamond\varphi]\!]^{\mathcal{K}}_{\rho} &= \{s \in S \mid \exists t \in S \text{ with } s \rightarrow t \text{ such that } t \in [\![\varphi]\!]^{\mathcal{K}}_{\rho}\}.
\end{aligned}$$

Standard propositional abbreviations and abbreviations like $\Box\varphi := \neg\Diamond\neg\varphi$ etc. are freely used. We will write $\mathcal{K}, s, \rho \models \varphi$ if $s \in [\![\varphi]\!]^{\mathcal{K}}_{\rho}$. For closed formulas, i.e. formulas with no free second-order variables, or if ρ is clear from the context we may simply write $\mathcal{K}, s \models \varphi$.

2.4.2 The Knaster-Tarski Fixed Point Theorem

To understand fixed points in the context of modal logics or later also in the context of hybrid logics it is helpful to recap a bit of mathematical theory about fixed points.

Definition 2.13. Let L be a set. A *partial order* on L – often denoted by $\leq$ – is a binary relation on L that is reflexive, antisymmetric and transitive. We say that $(L, \leq)$ is a partially ordered set.

Definition 2.14. Let $(L, \leq)$ be a partially ordered set and $\mathcal{S} \subseteq L$. An *upper bound* of S is an element $u \in L$ such that $s \leq u$ for all $s \in \mathcal{S}$. An upper bound u is called the *supremum* of K or $\sup(K)$ if $u \leq x$ holds for all upper bounds x of $\mathcal{S}$. Similarly a *lower bound* of K is an element $l \in L$ such that $l \leq s$ for all $s \in \mathcal{S}$ and a lower bound is called the infimum or $\inf(K)$ if it is the greatest upper bound, i.e. $x \leq l$ for all lower bounds x of K.

Definition 2.15. Let $(L, \leq)$ be a partially ordered set and $f : L \to L$ a function. Then $s \in L$ is called a *fixed point* if $f(s) = s$. In addition s is called a least (resp. greatest) fixed point if $s \leq t$ (resp. $s \geq t$) for all fixed points t.

Least and greatest fixed points as well as suprema and infima need not necessarily exist but are unique if they do.

Definition 2.16 (complete lattice)**.** A *complete lattice* is a partially ordered set $(L, \leq)$ such that every subset of L has a supremum and an infimum.

Example 2.17. Let $\mathcal{K} = \langle S, \to, L \rangle$ be a Kripke structure. Then $(2^S, \subseteq)$ – the powerset of the state space equipped with the subsetrelation – is a complete lattice. An illustration of this lattice is shown in Figure 2.1.
Let $\mathcal{S} \subseteq 2^S$ with $\mathcal{S} = \{S_1, \dots, S_m\}$ for some $m \in \mathbb{N}$. Then $\sup(\mathcal{S}) = \bigcup_{i=1}^m S_i$ and $\inf(\mathcal{S}) = \bigcap_{i=1}^m S_i$.

Definition 2.18. Let $f : X \to Y$ be a function between (partially) ordered sets. Then f is called *monotone* if for all $x, y \in X$ it holds that if $x \leq y$ then $f(x) \leq f(y)$.

Example 2.19. Let $\varphi(X)$ be a modal logic formula with a free second-order variable X such that X occurs only under the scope of an even number of negations and let $\mathcal{K} = \langle S, \to, L \rangle$ be a Kripke structure.
Then the function $f : 2^S \to 2^S$ with $V \mapsto [\![\varphi(X)]\!]^{\mathcal{K}}_{\rho[X \to V]}$ that maps a set V to the evaluation of $\varphi(X)$ when X is interpreted as the set V is a monotone function.
Thus, every modal formula with a free second-order variable under an even number of negations induces a monotone function.

The following theorem is the foundation for most fixed point constructions in mathematical logic and especially in the modal μ-calculus.

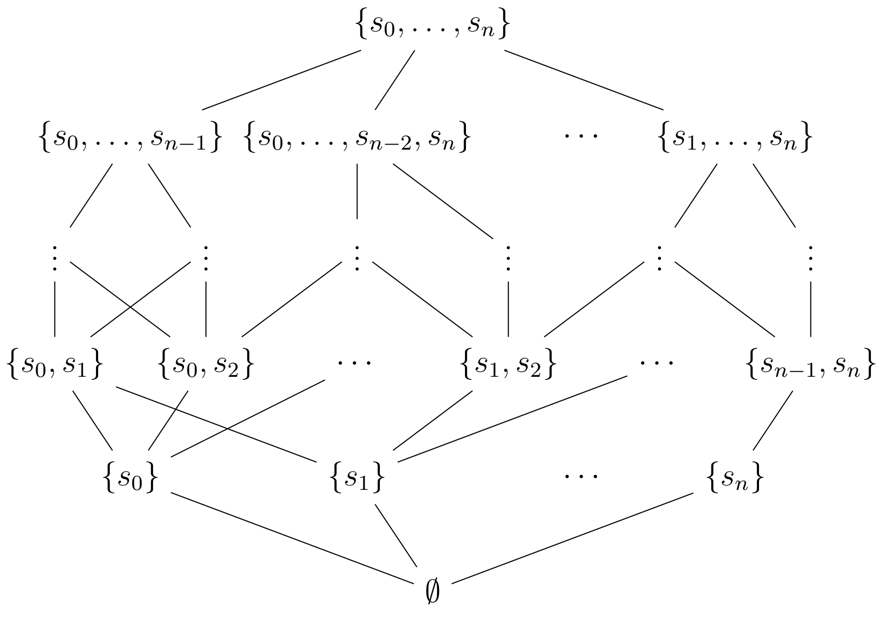


Figure 2.1: The powerset lattice over $S = \{s_0, s_1, \dots, s_n\}$.

Theorem 2.20 (Knaster-Tarski Fixed Point Theorem, [87])**.** Let $(L, \leq)$ be a complete lattice and $f : L \to L$ be a monotone function. Then the set P of all fixed points of f is not empty and $(P, \leq)$ is a complete lattice. In particular we have that the least fixed point can be characterised via

$$\mathit{lfp}(f) = \bigcap \{x \in L \mid f(x) \leq x\}$$

and the greatest fixed point is characterised through

$$\mathit{gfp}(f) = \bigcup \{x \in L \mid x \leq f(x)\}.$$

In other words, the Knaster-Tarski fixed point theorem guarantees the existence of least and greatest fixed points of monotone functions on complete lattices.

2.4.3 Syntax & Semantics of $\mathbf{L}_\mu$

We will use the Knaster-Tarski fixed point theorem over the powerset-lattice of Kripke structures combined with the monotone functions induced by a formula $\varphi(X)$ to give a well-defined semantics to fixed points in the context of modal logics.

Syntactically the *modal* μ-calculus – or short L_μ – simply adds least fixed point constructs to modal logic:

$$\varphi := p \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond\varphi \mid \mu X.\varphi(X).$$

To ensure a well-defined semantics we need the syntactic restriction that in fixed point formulas $\mu X.\varphi(X)$ all free occurences of X must be positive, i.e. under the scope of an even number of negations in φ. In this way φ induces a monotone operator which is guaranteed to have a least fixed point by the Knaster-Tarski theorem.
L_μ formulas are also interpreted with respect to a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ and an assignment for the free second-order variables $\rho : \mathit{Var}_2 \to 2^S$. The semantics of propositions, boolean connectives and diamonds are as in modal logic. To give meaning to the least fixed point formulas we use the characterisation of least fixed points from the Knaster-Tarski theorem:

$$[\![\mu X.\varphi(X)]\!]^{\mathcal{K}}_\rho = \bigcap\{T \subseteq S \mid [\![\varphi]\!]^{\mathcal{K}}_{\rho[X\to T]} \subseteq T\}.$$

We say that $X \in \mathit{Var}_2$ occurs *free* in $\varphi \in \mathrm{L}_\mu$ if there is no fixed point binder $\mu X.\psi(X)$ in the syntax tree above X in φ. Otherwise X is *bound*. Further, $\varphi[\psi/X]$ denotes the simultaneous substitution of every free occurrence of X by ψ.
Greatest fixed points can then simply be defined as a dual of smallest fixed points: $\nu X.\varphi(X) := \neg\mu X.\neg\varphi[\neg X/X]$, or equivalently also via the Knaster-Tarski theorem with the following interpretation:

$$[\![\nu X.\varphi(X)]\!]^{\mathcal{K}}_\rho = \bigcup\{T \subseteq S \mid T \subseteq [\![\varphi]\!]^{\mathcal{K}}_{\rho[X\to T]}\}.$$

If not stated otherwise we assume that only the least fixed point constructor is a native operator in the logic.
We assume that each $X \in \mathit{Var}_2$ is bound at most once by a fixed point quantifier μ or ν and denote by $\mathsf{fp}_\varphi(X)$ the function mapping each $X \in \mathit{Var}_2$ to its (unique) defining fixed point formula. If $\mathsf{fp}_\varphi(X) = \mu X.\psi(X)$ we say that X is of μ-type, otherwise X is of ν-type.

Example 2.21. The formula $\mu X.\, p \vee \Diamond X$ states that a state labelled p is reachable and is thus equivalent to the CTL formula $\mathsf{EF}p$.
The formula $\nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$ is a bit more involved. It states that there is a path on which p is seen infinitely often and is thus equivalent to the FCTL$^+$ formula $\mathsf{EGF}p$.

Definition 2.22. Let $\varphi \in \mathrm{L}_\mu$. We define $X \geq_\varphi Y$ if X has a free occurrence in $\mathsf{fp}_\varphi(Y)$ and we define $>_\varphi$ to denote the strict part of its transitive closure.

The *alternation depth of a μ-variable X* is the maximal length of a chain $X_1 \geq X_2 \geq \ldots \geq X_n$ where $X = X_1$, $X_1, X_3, \ldots$ are of μ-type and $X_2, X_4, \ldots$ are of ν-type. The *alternation depth of a ν-variable X* is defined analogously with the types of the descending variables in the chain switched.
The *alternation depth of φ* is the maximum of alternation depths of variables bound in φ.
For short we will write $\mathsf{ad}_\sigma(X)$ or $\mathsf{ad}(\varphi)$ to denote the alternation depth of a variable of type σ or the alternation depth of a formula. If σ is clear from the context we may also drop the index.

For example $\mathsf{ad}(\mu X.\, p \vee \Diamond X) = 1$ and $\mathsf{ad}(\nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X) = 2$.

2.4.4 Approximations

The characterisation of fixed points via Knaster-Tarski has theoretical appeal but is not very helpful in actually calculating the fixed points themselves. However, there is another equivalent characterisation of fixed points by Kleene [58] that gives rise to an iterative algorithm to approximate these fixed point formulas. Moreover, on certain "well-behaved" structures this algorithm even calculates the fixed points.

Definition 2.23. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure, ρ an assignment and $\mu X.\varphi(X)$ be a fixed point formula. The *approximations* of the fixed point with respect to ρ and $\mathcal{K}$ are defined as follows:

$$\begin{aligned}\mu X^0.\varphi(X) &:= \emptyset\\ \mu X^{(\alpha+1)}.\varphi(X) &:= [\![\varphi(X)]\!]^{\mathcal{K}}_{\rho[X\mapsto \mu X^\alpha.\varphi(X)]}\end{aligned}$$

for a successor ordinal α and

$$\mu X^\gamma.\varphi(X) := \bigcup_{\alpha<\gamma} \mu X^\alpha.\varphi(X)$$

for limit ordinals γ.
Let τ be the least ordinal such that $\mu X^\tau.\varphi(X) = \mu X^{(\tau+1)}.\varphi(X)$. Such an ordinal always exists. It is bounded by the cardinality of the Kripke structure. Then τ is called the *closure ordinal* of $\mathcal{K}$ and ρ with regard to $\mu X.\varphi(X)$.

Proposition 2.24. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure, ρ an assignment, $\mu X.\varphi(X) \in \mathrm{L}_\mu$ and τ the closure ordinal of $\mathcal{K}$ and ρ with regard to $\mu X.\varphi(X)$. Then it holds that

$$[\![\mu X.\varphi(X)]\!]^{\mathcal{K}}_\rho = \mu X^\tau.\varphi(X).$$

This is especially interesting on structures with finite closure ordinals – for example finite structures – because this opens up the possibility to iteratively calculate fixed points. We simply begin with the empty set and then evaluate $\varphi(X)$ with X mapping to the set of the previous iteration until the process eventually stabilises.
Another advantage of this iterative approach is that it can also help to understand formulas of the modal μ-calculus much more easily. The concise presentation of L_μ formulas comes at the cost of an easy-to-read notation. A *syntactic unfolding* of fixed point formulas that is based on these approximations can help to understand formulas much better. Such an unfolding is based on the following fact:

Proposition 2.25. For all μ-calculus formulas φ, ψ it holds that

$$[\![\varphi[\psi/X]]\!]^{\mathcal{K}}_{\rho} \quad = \quad [\![\varphi]\!]^{\mathcal{K}}_{\rho[X \mapsto [\![\psi]\!]^{\mathcal{K}}_{\rho}]} \ .$$

Thus, for finite ordinals α we can also syntactically unfold a fixed point formula $\mu X.\varphi(X)$ and represent the approximations by the following chain of formulas:

$$\varphi^0(X) := \mathtt{ff},$$
$$\varphi^{(\alpha+1)}(X) := \varphi[\varphi^\alpha/X].$$

A straightforward induction proves that $[\![\varphi^\alpha(X)]\!]^{\mathcal{K}}_{\rho} = \mu X^\alpha.\varphi(X)$.

Example 2.26. Consider the formula $\mu X.p \vee \Diamond X$. To understand this fixed point we take a look at its unfoldings:

$$\begin{aligned}
\varphi^0(X) &= \mathtt{ff},\\
\varphi^1(X) &= p \vee \Diamond \mathtt{ff},\\
\varphi^2(X) &= p \vee \Diamond(p \vee \Diamond \mathtt{ff}),\\
\varphi^3(X) &= p \vee \Diamond(p \vee \Diamond(p \vee \Diamond \mathtt{ff})),\\
\varphi^4(X) &= p \vee \Diamond(p \vee \Diamond(p \vee \Diamond(p \vee \Diamond \mathtt{ff}))),\\
&\vdots
\end{aligned}$$

which basically says either p holds now or p holds at a successor or p holds at a 2-step successor and so on. Thus the formula expresses reachability of a state that satisfies p.

2.5 Syntactic conventions

This section introduces some common terminology in the context of branching-time logics that will be used throughout this thesis. To give a unified definition of these basic concepts for every logic – including hybrid logics which will be defined later in Chapter 3 – we use the term *operator* as an abbreviation for any connective used to build formulas. The logics considered in this thesis exclusively feature *unary or binary operators*. For example, $\vee$ is a binary operator and E or $\neg$ are unary operators.
In order to analyse the computational complexity of these logics and their decision problems we define two measures of formula size.

Definition 2.27. Let φ be any branching-time formula. The set of subformulas $\mathsf{Sub}(\varphi)$ is defined recursively on the structure of φ by

$$\begin{aligned}
\mathsf{Sub}(p) &:= \{p\},\\
\mathsf{Sub}(\mathsf{O}'(\varphi_1)) &:= \{\mathsf{O}'(\varphi_1)\} \cup \mathsf{Sub}(\varphi_1),\\
\mathsf{Sub}(\mathsf{O}(\varphi_1,\varphi_2)) &:= \{\mathsf{O}(\varphi_1,\varphi_2)\} \cup \mathsf{Sub}(\varphi_1) \cup \mathsf{Sub}(\varphi_2),
\end{aligned}$$

for all unary operators O' and all binary operators O. The *size* of φ is defined by $|\varphi| := |\mathsf{Sub}(\varphi)|$.

Definition 2.28. The *length* of a formula φ is defined recursively on the structure of φ by

$$\begin{aligned}
\mathsf{length}(p) &:= 1,\\
\mathsf{length}(\mathsf{O}'(\varphi')) &:= 1 + \mathsf{length}(\varphi'),\\
\mathsf{length}(\mathsf{O}(\varphi_1,\varphi_2)) &:= 1 + \mathsf{length}(\varphi_1) + \mathsf{length}(\varphi_2)
\end{aligned}$$

for all unary operators O' and all binary operators O.

The length of a formula is at least as big as the size of the same formula and one can show that the length of a formula is at most exponentially longer than the size of the formula, c.f. [17].
For many proofs it is often easier if the formulas are shaped in a certain way. We say a formula $\varphi \in \mathrm{CTL}^*$ resp. $\varphi \in \mathrm{L}_\mu$ is in *negation normal form* if negation only occurs directly in front of atomic propositions. The following lemma is well-known and easily proven both for CTL^* as well as L_μ by pushing negations inwards and possibly using the dual temporal operators resp. greatest fixed points as first-class citizens as well as standard equivalences from propositional logic.

Lemma 2.29. For each formula $\varphi \in \text{CTL}^*$ an equivalent formula $\varphi' \in \text{CTL}^*$ in negation normal form can be computed in linear time. The same is true for L_μ.

The next two definitions are especially needed for CTL^* and its fragments. Similar concepts exist for L_μ but are not needed in this thesis.
In many situations involving branching-time logics the set of subformulas is not quite enough. Often we also need to deal with the unraveling of Until-formulas based on the equivalence $\varphi_1 \mathsf{U} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2))$ and thus we need to deal with an extended set of "subformulas“ that also includes the formula $\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)$. This slightly extended set is often referred to as the Fischer-Ladner closure [35] and is made precise in the following definition.

Definition 2.30. Let $\varphi \in \text{CTL}^*$. The *Fischer-Ladner closure* of φ is the smallest set $\mathsf{Fl}(\varphi)$ such that $\mathsf{Sub}(\varphi) \subseteq \mathsf{Fl}(\varphi)$ and that is closed under the following rule:

- For every formula $\varphi_1 \mathsf{U} \varphi_2 \in \mathsf{Fl}(\varphi)$ we also have $\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2) \in \mathsf{Fl}(\varphi)$.

It should be clear that the Fischer-Ladner closure of a CTL^* formula is finite. Especially its size is at most $2 \cdot |\varphi|$.
Another measure of branching-time formulas that is often used when comparing the expressive power of certain logics is the nesting depth of temporal operators.

Definition 2.31. Let $\varphi \in \text{CTL}^*$. We define the *temporal nesting depth* of φ – or short $\mathsf{nd}(\varphi)$ – as follows. For state formulas we have

$$\begin{aligned}
\mathsf{nd}(p) &:= 0, \\
\mathsf{nd}(\neg\varphi) &:= \mathsf{nd}(\varphi), \\
\mathsf{nd}(\varphi_1 \vee \varphi_2) &:= \max\{\mathsf{nd}(\varphi_1), \mathsf{nd}(\varphi_2)\}, \\
\mathsf{nd}(\mathsf{E}\psi) &:= \mathsf{nd}(\psi).
\end{aligned}$$

And for path formulas we have

$$\begin{aligned}
\mathsf{nd}(\neg\psi) &:= \mathsf{nd}(\psi), \\
\mathsf{nd}(\psi_1 \vee \psi_2) &:= \max\{\mathsf{nd}(\psi_1), \mathsf{nd}(\psi_2)\}, \\
\mathsf{nd}(\mathsf{X}\psi) &:= \mathsf{nd}(\psi) + 1, \\
\mathsf{nd}(\psi_1 \mathsf{U} \psi_2) &:= \max\{\mathsf{nd}(\psi_1), \mathsf{nd}(\psi_2)\} + 1.
\end{aligned}$$

The temporal nesting depth thus simply counts how many times X- and U-operators have been nested inside a formula.

Remark 2.32. Note, that often the nesting depth of a temporal formula is used to refer to the nesting depth of an LTL-like formula which in our context of branching-time logics means to a single path formula without counting nested state formulas. Our definition also counts the temporal nesting depth of embedded state formulas towards the temporal nesting depth. Thus, this definition also covers the case of counting the temporal nesting depth of an LTL-like formual.
We will need both options in this thesis. The full definition that also counts nested state formulas towards the nesting depth is especially needed for the Ehrenfeucht-Fraïssé Games for CTL presented in Section 5.1. The nesting depth of path formulas without counting nested state formulas is then needed in Section 5.2.2 where we explicitly introduce and work on "pure" path formulas, i.e. path formulas with no nested state formulas other than atomic ones which by definition have nesting depth 0.

Finally, by $\varphi[\psi_1/\chi_1, \ldots, \psi_n/\chi_n]$ we denote the simultaneous substitution of all subformulas χ_i by ψ_i for all $i \in \{1, \ldots, n\}$ with the exception of $\chi_i = X$ for some $X \in \mathit{Var}_2$, then we only substitute all free occurrences of X by ψ_i.

2.6 Expressiveness & A Branching-Time Hierarchy

A natural question that arises when studying different logics over the same type of structures is that of expressive power. What properties can be expressed in a certain logic? How do different logics relate to one another? Can one logic express properties that another cannot? And many more similar questions arise.
In this section we introduce a simple framework to compare the expressive power of different logics and then recall some important results comparing the previously introduced branching-time logics.

Definition 2.33. Let φ, χ be two branching-time formulas interpreted over a class $\mathbb{C}$ of structures. We say that φ and χ are *equivalent with respect to* $\mathbb{C}$ or short $\varphi \equiv_{\mathbb{C}} \chi$ if for all $\mathcal{K} \in \mathbb{C}$ and states $s \in S$ it holds that $\mathcal{K}, s \models \varphi$ if and only if $\mathcal{K}, s \models \chi$.
We also say that $\varphi \equiv \chi$ if φ, χ are equivalent over the class $\mathbb{K}$ of all Kripke structures. Moreover, we say that $\varphi \equiv_{\mathsf{fin}} \chi$ (resp. $\varphi \equiv_{\mathsf{tree}} \chi$) if φ, χ are equivalent over the classes of finite structures $\mathbb{F}$ (resp. the class of all trees $\mathbb{T}$).

Definition 2.34. Let Ψ, Ω be two logics interpreted over a class of structures $\mathbb{C}$. We say that Ψ *is at least as expressive as* Ω – or short $\Psi \succeq_{\mathbb{C}} \Omega$ – if and

only if for all formulas $\varphi \in \Omega$ there is a formula $\chi \in \Psi$ such that $\varphi \equiv_{\mathbb{C}} \chi$. Moreover, we say that $\Psi \equiv_{\mathbb{C}} \Omega$ if and only if $\Psi \succeq_{\mathbb{C}} \Omega$ and $\Omega \succeq_{\mathbb{C}} \Psi$. Moreover, we say that $\Psi \succ \Omega$ or Ψ *is more expressive than* Ω if and only if $\Psi \succeq \Omega$ but $\Omega \not\succeq \Psi$.
And finally we call Ψ and Ω *incomparable* if neither $\Psi \succeq \Omega$ nor $\Omega \succeq \Psi$ holds. In the case of all Kripke structures we usually drop the index $\mathbb{K}$ and simply write $\Psi \succeq \Omega$ or $\Psi \equiv \Omega$. Similarly to the previous definition in the case of finite structures or trees we also use the indices fin or tree.

Example 2.35. The branching-time logics from CTL to CTL* form a syntactical hierarchy. Thus, it is obvious by their definition that

$$\text{CTL} \preceq \text{CTL}^+ \preceq \text{FCTL}^+ \preceq \text{CTL}^*.$$

However, the other directions and the relationship to L_μ are not so obvious. However, the following relationships are quite well-known by now:

$$\text{CTL} \equiv \text{CTL}^+ \prec \text{FCTL}^+ \prec \text{CTL}^* \prec \text{L}_\mu.$$

The equivalence between CTL and CTL$^+$ is usually shown via a translation from CTL$^+$ to CTL [29]. The main difficulty in this translation are formulas of the form $\mathsf{E}(\bigwedge_{i\in I} \varphi_i \mathsf{U} \chi_i)$. Such Boolean combinations of until-formulas are not allowed in CTL. The trick of the translation is to "guess" an order in which all the until formulas are satisfied and then to recreate the path formula piece by piece. We will show a more general statement in Section 5.2 that also holds in the case of hybrid CTL and CTL$^+$.
The expressiveness gap between CTL and FCTL$^+$ can be shown by the formula $\mathsf{EGF}p$ (c.f. [30]) and the gap to CTL* can be shown for example by the formula $\mathsf{AF}(p \wedge \mathsf{X}p)$ which utilises full nesting of path formulas that is disallowed in all logics below CTL* (c.f. [30]). We also show a more general statement in Section 5.2 that proves that not even the hybrid extension of FCTL$^+$ can express this CTL* formula.
Lastly, it was shown in [23, 24] that L_μ is even more expressive than CTL*.

2.7 Model Checking

Generally the model checking problem is the following: Given a structure and a property decide whether this property holds on this structure. If the property holds, the structure is usually called a model of this particular property. Hence the term model checking.
In the context of (hybrid) temporal and modal logics there are usually two versions of this problem. The *local* model checking problem is the following:

Input: a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$, a state $s \in S$ and a (hybrid) temporal formula φ.

Output: Does $\mathcal{K}, s \models \varphi$ hold?

And the *global* model checking problem is the following:

Input: a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ and a (hybrid) temporal formula φ.

Output: The set of all states s such that $\mathcal{K}, s \models \varphi$ holds.

The complexity of model checking algorithms is usually measured in the worst-case space or time usage depending on the input size. The input is usually measured with respect to the Kripke structure and the formula. Hence the input size is $|\mathcal{K}| + |\varphi|$.
This may be sufficient for theoretical purposes. However these theoretical complexity bounds often do not match with the empirical data of practical applications. Thus, often a more refined analysis that separates both input sizes is needed. This corresponds to considering one of the input parts as fixed and measuring the complexity only in terms of either the size of the Kripke structure or the size of the formula. Such notions are known as *data complexity* – when the formula is fixed – and *expression complexity* – when the structure is fixed. Measuring the complexity with respect to both the structure and the formula is then referred to as the *combined complexity.*
Data complexity is usually important for practical purposes. In scenarios like evaluating database queries or verification of reactive systems the formula is often relatively small compared to the structure and one is generally more interested in the behaviour relative to large systems or databases.
Expression complexity is typically less important for practical scenarios. Instead it can be seen as an indicator of how (computationally) powerful a logic is.
We will often use the connection between data, expression and combined complexity. It is easy to see that upper bounds on the combined complexity transfer immediately to data and expression complexity. Also lower bounds in data or expression complexity also transfer to the combined complexity of an instance of the model checking problem.
Since the combined complexity is the standard notion for measuring complexity of the model checking problem we will mostly simply refer to it as *the complexity* of the model checking problem and explicitly mention when we refer to the data or expression complexity of a model checking problem.

2.8 Automata

Automata are devices that usually accept finite or infinite words or structures like trees and are often closely related to temporal logics. For example, most decidability results for branching-time logics are achieved by constructing a suitable tree automaton that can then be checked for emptiness.
Linear temporal logic and thus also the path formulas in the branching-time logics introduced earlier more closely relate to automata that accept infinite words like Büchi automata. We will introduce some of these devices in this section and present some problems and basic results about these automata that will be used in this thesis.

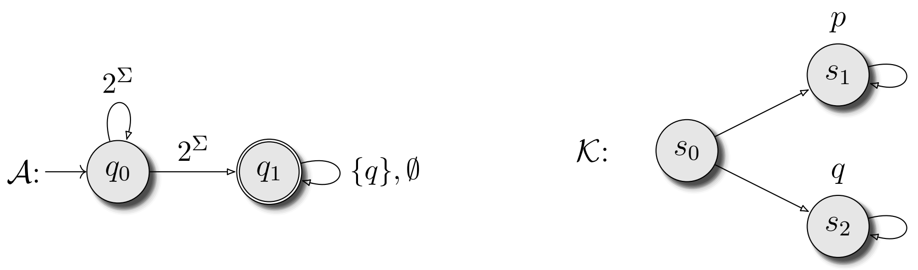
Definition 2.36. *A (nondeterministic) Büchi automaton* – or short NBA– is a tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, q_I, \delta, F)$ such that

- $\mathcal{Q}$ is a finite set,
- Σ is a finite alphabet,
- $q_I \in \mathcal{Q}$ is the initial state,
- $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a transition relation and
- $F \subseteq \mathcal{Q}$ are the accepting states.

A *run* of $\mathcal{A}$ on an infinite word $w = w_0 w_1 w_2 \ldots \in \Sigma^\omega$ is a sequence $\rho = (q_i)_{i \in \mathbb{N}}$ of states such that $q_0 = q_I$ and for all $i \in \mathbb{N}$ it holds that $(q_i, w_i, q_{i+1}) \in \delta$. By $\mathsf{Inf}(\rho)$ we denote the set of states q such that $q = q_i$ for infinitely many $i \in \mathbb{N}$. A run ρ is *accepting* if $\mathsf{Inf}(\rho) \cap F \neq \emptyset$.
The language of $\mathcal{A}$ is defined as the set of words $w \in \Sigma^\omega$ such that there exists an accepting run for w on $\mathcal{A}$.
An NBA $\mathcal{A}$ is called *deterministic* or a DBA if for every $q \in \mathcal{Q}$ and every $a \in \Sigma$ there is exactly one state $q' \in \mathcal{Q}$ such that $(q, a, q') \in \delta$.

Let $\mathcal{K} = \langle S, \rightarrow, L \rangle$ be a Kripke structure over some set of atomic propositions *Prop* and let $\mathcal{A}$ be a Büchi automaton over $\Sigma = 2^{Prop}$. We say that a path π in $\mathcal{K}$ is accepted by $\mathcal{A}$ iff $\mathcal{A}$ accepts the word $(l_i)_{i \in \mathbb{N}}$ with $l_i \coloneqq \{p \in Prop \mid \pi^i \in L(p)\}$.

Example 2.37. Let $Prop = \{p, q\}$ and $\mathcal{A}$ be the NBA over $\Sigma = 2^{Prop}$ depicted in Figure 2.2a. It is easy to see that $\mathcal{A}$ accepts all paths that eventually do not read any more p's.
Thus, if we look at the structure $\mathcal{K}$ in Figure 2.2b we can see that the path $s_0(s_2)^\omega$ is accepted but the path $s_0(s_1)^\omega$ is not.



(a) NBA $\mathcal{A}$. (b) Kripke Structure $\mathcal{K}$.

Figure 2.2: Structure $\mathcal{A}$ and a sketch of Duplicator's path choice on $\mathcal{A}$.

The *existential* NBA *path problem* is the following:

Input: a Kripke structure $\mathcal{K} = \langle S, \to, L \rangle$, a state $s \in S$ and a Büchi-automaton $\mathcal{A}$.

Output: Is there a path π in $\mathcal{K}$ starting at s such that $\mathcal{A}$ accepts π?

It is known that this problem can be solved efficiently using two nested depth-first searches.

Proposition 2.38 ([22])**.** The existential NBA path problem can be solved in NLogSpace.

Parity Automata. The class of languages accepted by an NBA form the so called ω-regular word languages. However, sometimes it may be hard to use Büchi-automata directly and seemingly "stronger" acceptance conditions are easier to use.

Definition 2.39. *A (nondeterministic) parity automaton* – or short NPA– is a tuple $(\mathcal{Q}, \Sigma, q_I, \delta, \Omega)$ with $\mathcal{Q}, \Sigma, q_I, \delta$ defined as for NBA and $\Omega : \mathcal{Q} \to \mathbb{N}$ is a priority function.
We say that a parity automaton $\mathcal{P} = (\mathcal{Q}, \Sigma, q_I, \delta, \Omega)$ has *index k* if $|\{\Omega(q) \mid q \in \mathcal{Q}\}| = k$.
Runs are defined as for NBA. A run ρ on an NPA is called *accepting* if and only if $\max\{\Omega(q) \mid q \in \mathsf{Inf}(\rho)\}$ is even, i.e. if the maximal priority that occurs infinitely often on this run is even. Deterministic parity automata or DPA are defined in the same way as DBA.

It is easily seen that for each NBA there is an equivalent NPA: NBA can be seen as special parity automata with only two priorities, e.g. $\Omega(q) = 2$ if $q \in F$ and $\Omega(q) = 1$ if $q \notin F$. However, it is also well-known that each NPA can also be translated into an equivalent NBA [47]. Thus, parity automata and Büchi automata accept the same class of languages, called the ω-regular languages.

2.9 Games

In the field of modal and temporal logics 2-player games often present a nice framework that helps to reason about certain aspects of these logics. Often times decision problems like model checking can be reduced to solving a 2-player game. A prominent example are the model checking games for the μ-calculus that we will present in Section 2.9.3. Another example are Ehrenfeucht-Fraïssé games which are used to reason about the limits of the expressive power of a logic.
We use this section to introduce a general framework of 2-player games and to introduce the terminology associated with these games. Special instances of 2-player games will be used throughout this thesis, especially model checking games and Ehrenfeucht-Fraïssé games. We will also take a look at the important class of parity games and state some important properties.

2.9.1 A Framework for 2-Player Games

A 2-player game is – as the name suggests – played between two players, here named 0 and 1, and consists of a *game arena*, a graph, and a *winning condition.* It usually can be thought of as two players pushing a token along the edges of the game arena and each player naturally tries to win the game. To better distinguish both players we usually refer to player 0 with female pronouns and to player 1 with male ones.

Definition 2.40. The *game arena* of a 2-player game is a tuple (V, V_0, V_1, E) where V is a (usually finite) set of states or nodes, $E \subseteq V \times V$ is an edge relation and V_0, V_1 form a partition of V. We say that a node $v \in V$ *belongs* to player i if $v \in V_i$.
A finite sequence $\lambda = v_0 v_1 v_2 \dots v_n$ such that $(v_i, v_{i+1}) \in E$ for every $0 \leq i < n$ is called a *partial play* and a partial play λ is called *maximal* if there is no partial play λ' such that λ is a proper prefix of λ'.
A *play* λ is a maximal partial play or an infinite sequence $v_0 v_1 v_2 \dots$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \mathbb{N}$. We will say that λ *starts* in v_0. Further, we denote the set of all infinite plays by Π and the set of all finite plays by Π_{fin}.

We will usually not strictly differentiate between a partial play and a play. It should be clear from the context which of the two is meant.

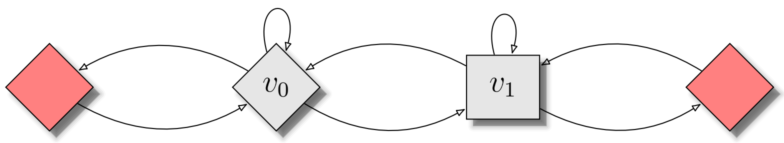


Figure 2.3: The game arena for the game $\mathcal{G}$.

Definition 2.41. A *2-player game* or simply just a *game* is a tuple $\mathcal{G} = (V, V_0, V_1, E, v_I, F)$ where (V, V_0, V_1, E) is a game arena, $v_I \in V$ is an initial state and $F : \Pi \cup \Pi_{\mathsf{fin}} \to \{0, 1\}$ is a *winning condition* that assigns a winner to each play starting at the initial state.
A game is played by pushing a *game token* along the edges of the game arena starting at some initial state $v_I \in V$. Depending on wether the node that currently has the token belongs to player 0 or 1 it is this player's choice to choose an outgoing edge from this node and move the token along one of these edges to another node of the arena. The game continues either until the token gets stuck, i.e. there are no outgoing edges to choose, or it continues forever. In both cases a play λ is formed. The play is won by player i if and only if $F(\lambda) = i$.

Example 2.42. Take a look at the game $\mathcal{G} = (V, V_0, V_1, E, v_0, F)$ whose game arena is depicted in Figure 2.3. Nodes that belong to player 0 have a diamond shape while nodes belonging to player 1 have a rectangular shape. The winning condition F is that a play is won by player 0 if and only if there is a red node that appears infinitely often in the play. Since the game arena is total, i.e. there is an outgoing edge on every node we do not need to consider finite plays.

The winner of a single play is easily obtained by simply checking the winning condition. However, the basic question when it comes to 2-player games is usually a bit more general: Can one of the players play in such a way that he *surely* wins the game, no matter how the opposing player chooses to play? This question is often hard to answer since every move needs to consider all possible choices of the other player. This question is formalised with the notion of *strategies*.

Definition 2.43. Let $\mathcal{G} = (V, V_0, V_1, E, v_I, F)$ be a game and $\lambda = v_0 v_1 \ldots v_n$ be a finite play on $\mathcal{G}$. We say that λ is owned by player i if $v_n \in V_i$. Let $\Pi_{\mathsf{par},i}$ denote the set of all (partial) plays owned by player i.
A strategy for player i is a partial function $\mathsf{str} : \Pi_{\mathsf{par},i} \to V$ which assigns to every (partial) play $v_0 v_1 \ldots v_n$ owned by player i a new node $w \in V$ such that $(v_n, w) \in E$. We say that a (finite or infinite) play $\lambda = v_0 v_1 \ldots$

conforms to the strategy str if for every $j \in \mathbb{N}$ with $v_j \in V_i$ it holds that $v_{j+1} = \mathsf{str}(v_0 \dots v_j)$.
A winning strategy for player i is a strategy such that every play that conforms to this strategy is won by player i.

Thus, the question if one of the players can surely win a game can be reformulated into: Does a winning strategy for one of the players exist?
We often refer to the decision problem "Is there a winning strategy for player 0?“ as *solving the game*.

Example 2.44. Look again at the game $\mathcal{G}$ from Example 2.42. If the game starts at v_0 then player 0 has a winning strategy. She simply moves to the leftmost red-marked state and then continues to go back to v_0 where he can again go to the leftmost state and so on. In this way player 0 surely sees a red state infinitely often.
On the other hand, if the game starts for example at v_1 then player 1 has a winning strategy. He can simply loop back to v_1 and never leave this state to win the game. In fact, he can also visit the rightmost red state a number of times because at this state player 0 has no choice but to move the token back to v_1 again. As long as player 1 at some point decides to never leave v_1 again he wins.

Keep in mind that these definitions only provide a rough framework and terminology for 2-player games. We will sometimes deviate slightly from this framework. For example, we will sometimes not explicitly partition the set of nodes if the edge relation from some node is deterministic. Or a single move might involve more than one step in a play. This is for example the case in the Ehrenfeucht-Fraïssé games described in Section 5.1.

2.9.2 Parity Games & other Regular Games

Parity games, just as parity automata, are a class of 2-player games that is intensively studied because of their multitude of applications in automata theory, logic, verification and synthesis. One particular reason is their close connection to the model checking problem for L_μ.

Definition 2.45. A *parity game* is a 2-player game $\mathcal{G} = (V, V_0, V_1, E, v_I, \Omega)$ where the winning condition is given via a function $\Omega : V \to \mathbb{N}$ that assigns a priority to each state.
Typically parity games are assumed to be total, i.e. there is an edge from every state and thus there are only infinite plays in such a game.

A play $\lambda = v_0 v_1 \dots$ is won by player 0 if and only if $\max\{\Omega(v) \mid v \in \mathsf{Inf}(\lambda)\}$ is even with $\mathsf{Inf}(\lambda)$ denoting the set of all states occurring infinitely often in λ.
Similar to parity automata we refer to the number of different priorities as the index of $\mathcal{G}$.

Example 2.46. In fact, the game $\mathcal{G}$ presented in Example 2.42 can also be seen as a parity game. For this, assign the red states the priority 2 and the other states only priority 1. It is not difficult to see that the winning condition then matches the previous one.

Parity games have some nice theoretical properties: All parity games *determined*, i.e. there is always a winning strategy for one of the players [31]. Moreover, it can be shown that these winning strategies are *positional* which means that such a strategy only needs to consider the current state on which the token lies and not the whole history on how the token got to this place as it is generally the case for two-player games [31].
Similarly to the connection between NPA and NBA (or other automaton models for ω-regular languages) it is also possible to reduce 2-player games in which the winning condition is regular, i.e. it is given by an ω-regular automaton, to parity games.
We first describe what it means that the winning condition of a 2-player game is described by an automaton.

Definition 2.47. Let $\mathcal{G} = (V, V_0, V_1, E, v_I, F)$ be a 2-player game and Σ be a finite set of *symbolic rule names*. A symbolic representation of the game rules is a function $f : E \to \Sigma$ such that every edge in the game is linked to a symbolic rule application.
We can thus naturally associate each play $\lambda = v_0 v_1 \dots$ with its symbolic play $f(\lambda) := f(v_o, v_1) f(v_1, v_2) \dots$ and denote by $f(\mathcal{W}_0)$ resp. $f(\mathcal{W}_1)$ the set of all symbolic plays won by player 0 resp. 1.
We only consider symbolic representations such that if $f(\lambda) = f(\lambda')$ for two plays λ, λ' then λ is won by player 0 if and only if λ' is won by player 0 and only consider them for determined games.
We say that the winning condition is *regular* if there is an ω-regular automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = f(\mathcal{W}_0)$ and refer to games with regular winning conditions as *regular games*.

For $\Sigma = E$ and f the identity function we simply get that the winning condition is recognised by an ω-regular automaton over the set of edges in the game. However, using proper symbolic encdings where Σ may be much smaller than the set of edges in a particular type of game may result in smaller automata and thus better complexity bounds.

Regular games can be reduced to parity games with an easy product construction enabling the use of one of the many solvers for parity games [39, 90]. However, for this construction it is essential that the winning condition is given by a *deterministic* automaton (c.f. [24, Sect. 15.1.7]). This means that potential nondeterministic automata for the winning conditions need to be determinised before they can be used. This of course comes at a cost and needs to be considered.

Proposition 2.48 ([24])**.** Let $\mathcal{G} = (V, V_0, V_1, E, v_I, \Omega)$ be a 2-player game with n nodes such that its winning condition is recognised by a deterministic parity automaton with m states and index k. Then there is a parity game $\mathcal{G}'$ with at most $m \cdot n$ nodes and index k that is won by player 0 if and only if $\mathcal{G}$ is won by player 0.

Solving parity games is known to be in NP $\cap$ Co-NP [48], however most practically used algorithms are still exponential in the number of priorities. A class of parity games that is easier to solve are 1-player parity games.

Definition 2.49. Let $\mathcal{G} = (V, V_0, V_1, E, v_I, \Omega)$ be a parity game. $\mathcal{G}$ is called a 1-*player parity game* if $V_0 = V$ or $V_1 = V$.

1-player parity games are special parity games where all nodes belong to one of the players and thus this player is the only one determining the outcome. It is not surprising that these games are very easy to solve since these games basically boil down to solving a reachability problem.

Proposition 2.50 ([24])**.** 1-player parity games can be solved in NLog-Space.

A great example that shows why parity games are particularly useful tools are the model checking games for the modal μ-calculus.

2.9.3 Model Checking Games for L_μ

Model checking games for the modal μ-calculus [85] are a great way to not only solve the model checking problem for L_μ but also help to understand what exactly a possibly convoluted μ-calculus formula expresses because they unfold a formula step by step. We will present these well-known games and give a short example. For a more detailed introduction see for example [85, 16].

Definition 2.51. Let $\varphi \in \mathrm{L}_\mu$ be in negation normal form and $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure. The L_μ model checking game $\mathcal{G}_{\mathrm{L}_\mu}(\mathcal{K}, \varphi)$ is played between two players $\boldsymbol{V}$ and $\boldsymbol{R}$ called the Verifier and the Refuter.

$$\begin{array}{ccc}
\dfrac{s \vdash \psi_1 \wedge \psi_2}{s \vdash \psi_1 \quad s \vdash \psi_2}\ (\boldsymbol{R}) & \dfrac{s \vdash X}{s \vdash \mathsf{fp}_\varphi(X)} & (\boldsymbol{V})\ \dfrac{s \vdash \psi_1 \vee \psi_2}{s \vdash \psi_1 \quad s \vdash \psi_2} \\
\dfrac{s \vdash \Box\psi}{t \vdash \psi}\ (\boldsymbol{R} : s \to t) & \dfrac{s \vdash p}{s \vdash p} & (\boldsymbol{V} : s \to t)\ \dfrac{s \vdash \Diamond\psi}{t \vdash \psi} \\
\dfrac{s \vdash \nu X.\psi(X)}{s \vdash \psi(X)}\ (\boldsymbol{R}) & \dfrac{s \vdash \neg p}{s \vdash \neg p} & (\boldsymbol{V})\ \dfrac{s \vdash \mu X.\psi(X)}{s \vdash \psi(X)}
\end{array}$$

Figure 2.4: The game rules for the L_μ model checking games.

The game arena is the set of configurations $S \times \mathsf{Sub}(\varphi)$. We usually denote a configuration via $s \vdash \psi$. The game begins at some position $s \vdash \varphi$ and evolves using the rules depicted in Figure 2.4. These rules define the partition of the game arena into nodes belonging to $\boldsymbol{V}$ and nodes belonging to $\boldsymbol{R}$ as well as the edge relation between the configurations. For example the rule

$$\frac{s \vdash \Box\psi}{t \vdash \psi}\ (\boldsymbol{R} : s \to t)$$

means that a configuration of the form $s \vdash \Box\psi$ belongs to $\boldsymbol{R}$ and he can choose a successor t of s in the Kripke structure and move the token to the configuration $t \vdash \psi$.
Intuitively $\boldsymbol{V}$ wants to prove that the structure is a model of the formula while $\boldsymbol{R}$ tries to refute that.
There are two ways a play can go. Either it gets stuck in a configuration $s \vdash p$ or it continues to unfold fixed points since these are the only rules that do not necessarily reduce the formula size in a configuration. For the first case $\boldsymbol{V}$ wins if $\mathcal{K}, s \models p$ and $\boldsymbol{R}$ wins if this is not the case.
The winner of an infinite play that keeps unfolding fixed points is intuitively determined by the type of the unique largest fixed point with respect to the dependency order of fixed points $>_\varphi$ occurring infinitely often. If it is of type ν then $\boldsymbol{V}$ wins and if it is of type μ then $\boldsymbol{R}$ wins.
Both cases can easily be formulated as a parity winning condition. For this we need to assign even priorities to ν-type variables and odd priorities to μ-type variables and we also need to make sure that if $X >_\varphi Y$ then $\Omega(X) > \Omega(Y)$. The latter condition can easily be formulated using the notion of alternation depth.
Further, we assign an even priority to configurations $s \vdash p$ if $s \in L(p)$ and an odd priority if this is not the case. Note that in these cases the game is stuck in this configuration and thus the exact value of the priority is not important. We will simply use priorities 0 and 1.

For example, one way to do this is to assign the following priorities to configurations α:

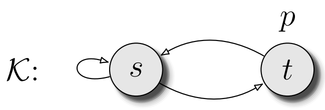
$$\begin{array}{ll}
\Omega(\alpha) = 2 \cdot \lfloor \mathsf{ad}(X)/2 \rfloor + 1, & \text{if } \alpha = s \vdash \mu X.\psi(X), \\
\Omega(\alpha) = 2 \cdot \lfloor \mathsf{ad}(X)/2 \rfloor, & \text{if } \alpha = s \vdash \nu X.\psi(X), \\
\Omega(\alpha) = 1, & \text{if } \alpha = s \vdash p \text{ and } s \notin L(p), \\
\Omega(\alpha) = 1, & \text{if } \alpha = s \vdash \neg p \text{ and } s \in L(p), \\
\Omega(\alpha) = 0, & \text{if } \alpha = s \vdash p \text{ and } s \in L(p), \\
\Omega(\alpha) = 0, & \text{if } \alpha = s \vdash \neg p \text{ and } s \notin L(p), \\
\Omega(\alpha) = 0, & \text{otherwise.}
\end{array}$$

The following fact about these model checking games is well-known for L_μ.

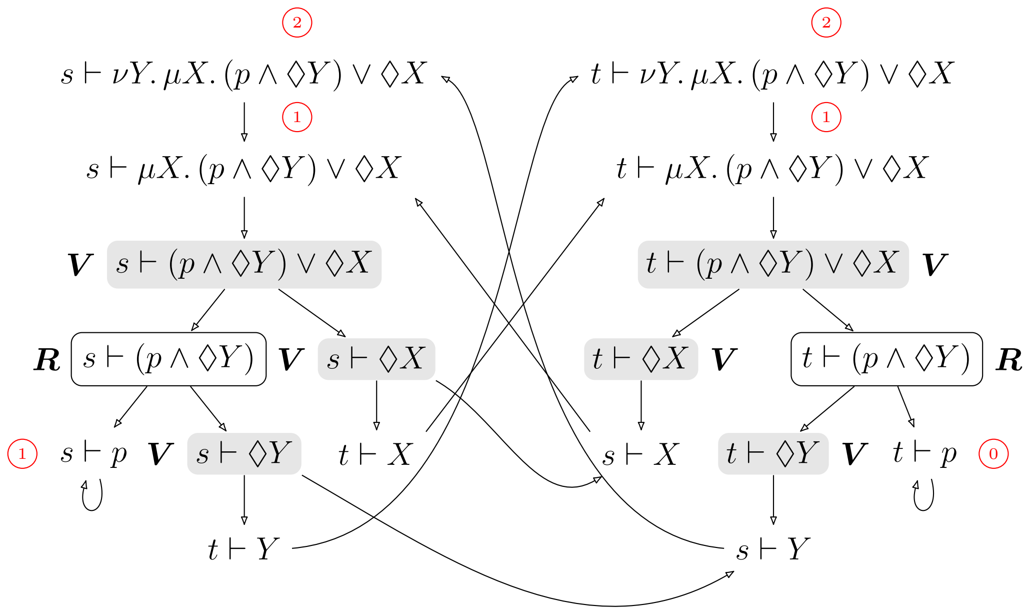
Proposition 2.52 ([85])**.** Player $\boldsymbol{V}$ has a winning strategy in $\mathcal{G}_{\mathrm{L}_\mu}(\mathcal{K}, \varphi)$ from $s \vdash \varphi$ if and only if $\mathcal{K}, s \models \varphi$.

We will use an example to showcase the use of these model checking games.

Example 2.53. Consider again the formula $\nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$ introduced in Example 2.21 and the Kripke structure



The corresponding model checking game $\mathcal{G}_{\mathrm{L}_\mu}(\mathcal{K}, \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X)$ is depicted in Figure 2.5.
Configurations belonging to $\boldsymbol{V}$ are marked grey while configurations belonging to $\boldsymbol{R}$ have marked borders. The numbers depicted in red are the priorities assigned to these configurations. If a priority is not depicted then it is 0.
We already know that the formula expresses the property that there is a path on which p occurs infinitely often. This is certainly the case on $\mathcal{K}$ with a path that infinitely often goes through t. Thus, $\boldsymbol{V}$ should have a winning strategy in this game, regardless of whether the game starts at state s or t.
First, we observe that $\boldsymbol{V}$ has a choice to make at up to six configurations in the game. However, the configurations $t \vdash \Diamond X$ and $t \vdash \Diamond Y$ are deterministic in that there is no real choice. Suppose for a moment that the game starts at $s \vdash \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$. Then the first choice for $\boldsymbol{V}$ happens at $s \vdash (p \wedge \Diamond Y) \vee \Diamond X$. She immediately loses if she chooses to go to $\boldsymbol{R}$'s configuration $s \vdash (p \wedge \Diamond Y)$ because p does not hold at s. Thus, $\boldsymbol{V}$ chooses to go to $s \vdash \Diamond X$.



2
$s \vdash \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$
1
$s \vdash \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$
$\boldsymbol{V}$ $s \vdash (p \wedge \Diamond Y) \vee \Diamond X$
$\boldsymbol{R}$ $s \vdash (p \wedge \Diamond Y)$
$\boldsymbol{V}$ $s \vdash \Diamond X$
1 $s \vdash p$
$\boldsymbol{V}$ $s \vdash \Diamond Y$
$t \vdash X$
$t \vdash Y$
2
$t \vdash \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$
1
$t \vdash \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$
$t \vdash (p \wedge \Diamond Y) \vee \Diamond X$ $\boldsymbol{V}$
$t \vdash \Diamond X$ $\boldsymbol{V}$
$t \vdash (p \wedge \Diamond Y)$ $\boldsymbol{R}$
$s \vdash X$
$t \vdash \Diamond Y$ $\boldsymbol{V}$
$t \vdash p$ 0
$s \vdash Y$

Figure 2.5: The game $\mathcal{G}_{\mathrm{L}_\mu}(\mathcal{K}, \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X)$.

Next, if she chooses to always go to $s \vdash X$ at $s \vdash \Diamond X$ then the game – with these two choices only – gets stuck in an infinite loop with the highest priority 1 by unfolding the fixed point formula of X again and again. Thus, $\boldsymbol{V}$ chooses the other alternative and goes to $t \vdash X$.
The next choice for $\boldsymbol{V}$ is then at $t \vdash (p \wedge \Diamond Y) \vee \Diamond X$. Here she can choose to go to $t \vdash (p \wedge \Diamond Y)$ because p holds at t and thus $\boldsymbol{R}$ can only choose to go to $t \vdash \Diamond Y$ if he does not want to lose immediately. This leads the play to go through $s \vdash \nu Y.\, \mu X.\, (p \wedge \Diamond Y) \vee \Diamond X$ again and which has priority 2 creating an infinite loop with the highest priority 2. Her choice at $s \vdash \Diamond Y$ is irrelevant since this configuration will never be reached.
Thus, $\boldsymbol{V}$ wins if she chooses to follow this strategy. If we look at this strategy carefully, we see that $\boldsymbol{V}$ wins because she forces the game to go through a state satisfying p infinitely often which is exactly what the formula expresses.

2.10 Computational Complexity

In this section we introduce the basic ideas and concepts of computational complexity. For a more in-depth introduction to the topic we refer to [76] or [9].

Oftentimes when dealing with logics one is not only interested in their *decision problems*, i.e. is there a model that satisfies a given formula, or does this particular system satisfy a given formula, but also in how costly these decision procedures are.
A standard way of measuring such algorithms is to measure the *time* and *space* that they use relative to the input size and then to classify the algorithms. The standard model of classification are *Turing machines*. They have proven to be a very robust mathematical model of computation and are by the Church-Turing thesis conjectured to be universal, i.e. every computable function is conjectured to be computable by a Turing machine. So far, this conjecture seems to hold.
Decision problems such as the model checking problem or the satisfiability problem of a logic in this context are usually considered as a set – for example the set of formulas that are satisfiable or the class of structures and formulas such that the structure is a model of the formula etc.
The basic model of a Turing machine consists of a finite control or a finite amount of internal states, an input tape which is divided into single cells and a tape head that is used to read or write on a single cell of the tape. Each cell contains at most one symbol from a predefined alphabet and the tape is considered to be infinite in both directions.
The tape is used for the input into the Turing machine, potential output, as well as a working memory. Given some input on the tape the Turing machine starts to read the contents of the first cell on the tape and consults its finite control which consists of a finite set of states and rules that tell it what to do when reading a specific symbol in a specific state. The machine can only rewrite the symbol on the tape, change to a different internal state and then move its head to one of the adjacent cells on the tape. The computation is finished if the machine (potentially after several computational steps) changes into a special final state.
Formally, such a machine is defined as follows.

Definition 2.54. A (deterministic) *Turing machine* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, N, R\}$ is a (partial) transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of accepting states.
A *configuration* of a Turing machine is a triple $(\alpha_1, q, \alpha_2) \in \Sigma^* \times Q \times \Sigma^*$ where $\alpha_1\alpha_2$ is the actual content of the tape and $q \in Q$ is the current state of the Turing machine.
Suppose that the Turing machine is in some configuration (α_1, q, α_2) and suppose that $\alpha_2 = x\beta$ for some $x \in \Sigma$. The Turing machine operates by reading x in state q and moving according to $\delta(q, x)$. For example, suppose

that $\delta(q,x) = (q', y, R)$. Then the machine writes the symbol y into the tape cell, changes its state to q' and moves the head to the right, reaching a configuration $(\alpha_1 y, q', \beta)$. The machine acts accordingly if the transition function dictates to move to the left (L) or to stay in the same position (N).
A *computation* of a Turing machine on some word $w \in \Sigma^*$ is a (finite or infinite) sequence of computational steps starting in the *initial configuration* (ε, q_0, w). If the Turing machine enters a configuration that has no following state, i.e. δ is undefined for it, then we say that the machine *halts*. A computation is called *accepting* if it is finite and ends with a configuration $(\alpha_1, q_f, \alpha_2)$ such that $q_f \in F$. The *language* of a Turing machine is the set of words $w \in \Sigma^*$ such that there exists an accepting computation for w.
A *nondeterministic* Turing machine is defined like a deterministic Turing machine with the exception that we have a transition relation $\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{L, N, R\}$ and computational steps only need to respect *some* tuple in this transition relation.

To classify problems we also need some kind of measure on Turing machines. This is measured in terms of their time and space usage for their worst-case acceptance behaviour.
Let M be a deterministic (nondeterministic) Turing machine and let $t : \mathbb{N} \to \mathbb{N}$ be a function. M is said to be *(nondeterministically)* $t(n)$*-time bounded* if for every input word of length n the Turing machine M makes at most $t(n)$ steps before halting. The language of M is then said to be of *(nondeterministic) time complexity* $t(n)$.
Likewise, if for every input word of length n the Turing machine visits at most $t(n)$ distinct cells before it halts, then we say that M is *(nondeterministically)* $t(n)$*-space bounded* and the language of M is of *(nondeterministic) space complexity* $t(n)$.
The family of languages of (nondeterministic) time complexity $\mathcal{O}(t(n))$ is defined as DTIME$(t(n))$ (NTIME$(t(n))$) and the family of languages of (nondeterministic) space complexity $\mathcal{O}(t(n))$ is defined as DSPACE$(t(n))$ (NSPACE$(t(n))$).

Definition 2.55. Let $2^n_0 \coloneqq n$ and $2^n_{m+1} \coloneqq 2^{2^n_m}$. We define the complexity classes as follows:

$$\begin{aligned}
\text{NLogSpace} &\coloneqq \text{NSPACE}(\log n),\\
\text{P} &\coloneqq \cup_{k\geq 1}\text{DTIME}(n^k),\\
\text{PSpace} &\coloneqq \cup_{k\geq 1}\text{DSPACE}(n^k),\\
\text{NPSpace} &\coloneqq \cup_{k\geq 1}\text{NSPACE}(n^k),\\
\text{NP} &\coloneqq \cup_{k\geq 1}\text{NTIME}(n^k),
\end{aligned}$$

$$\begin{aligned}
\textsc{ExpTime} &\coloneqq \cup_{k\geq 1}\mathrm{DTIME}(2^{n^k}),\\
\textsc{NExpTime} &\coloneqq \cup_{k\geq 1}\mathrm{NTIME}(2^{n^k}),\\
\textsc{ExpSpace} &\coloneqq \cup_{k\geq 1}\mathrm{DSPACE}(2^{n^k}),\\
\textsc{NExpSpace} &\coloneqq \cup_{k\geq 1}\mathrm{NSPACE}(2^{n^k}),\\
m-\textsc{ExpTime} &\coloneqq \cup_{k\geq 1}\mathrm{DTIME}(2^{n^k}_m),\\
m-\textsc{ExpSpace} &\coloneqq \cup_{k\geq 1}\mathrm{DSPACE}(2^{n^k}_m),\\
\textsc{Elementary} &\coloneqq \cup_{m\geq 1} m-\textsc{ExpSpace},\\
\textsc{Tower} &\coloneqq \mathrm{DSPACE}(2^n_n).
\end{aligned}$$

The following relationships between these complexity classes are known:

$$\begin{aligned}
&\textsc{NLogSpace} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \textsc{PSpace}\\
&\qquad\qquad \subseteq \textsc{ExpTime} \subseteq \textsc{NExpTime} \subseteq \textsc{ExpSpace}.
\end{aligned}$$

We also know that P $\subsetneq$ ExpTime [76]. However, it is not known if any of the direct subset relationships above are strict. By Savitch's Theorem [81] we get that PSpace = NPSpace as well as ExpSpace = NExpSpace.
For any of these complexity classes L we also define Co-L as the set of all problems whose complement is in L.
Lastly, we also define $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$, as the class of all problems solvable by a deterministic polynomial-time Turing machine with access to a one-step NP-oracle. We have NP $\cup$ Co-NP $\subseteq \Delta_2^p \subseteq$ PSpace. For a more detailed introduction to this complexity class, Oracle-Turing machines and the polynomial-time hierarchy, we refer to [86].
To further characterise how difficult a decision problem can be, it is also useful to compare decision problems relative to each other.

Definition 2.56. Let $\mathcal{P}, \mathcal{P}' \subseteq \Sigma^*$ be decision problems over some alphabet Σ. We say that $\mathcal{P}$ is polynomial-time-reducible (logspace-reducible) to $\mathcal{P}'$ if there is a function $r : \Sigma^* \to \Sigma^*$ such that

- for every word $x \in \Sigma^*$ it holds that $x \in \mathcal{P}$ if and only if $r(x) \in \mathcal{P}'$ and
- r can be computed in polynomial time (logarithmic space).

In this case r is also called a polynomial-time (logspace-)reduction from $\mathcal{P}$ to $\mathcal{P}'$.

Definition 2.57. Let $\mathcal{P} \subseteq \Sigma^*$ be a decision problem and L be a class of problems. We say that $\mathcal{P}$ is L-*hard* (with respect to polynomial time reductions) if for every problem $\mathcal{P}' \in$ L there is a polynomial-time reduction r from $\mathcal{P}$ to $\mathcal{P}'$.

Further, if $\mathcal{P}$ is L-hard *and* $\mathcal{P} \in$ L, then $\mathcal{P}$ is said to be L-complete (with respect to polynomial-time reductions).

Thus, if a problem is hard for a complexity class it can be considered to be as hard as any problem in this class and if it is complete for this class it can be considered to be "the hardest" problem in this class.

Representations. Turing Machines are an ideal tool to solve problems that require manipulation of strings. However, it is not quite clear by definition how Turing machines can be utilised for algorithms that require manipulation of other mathematical objects like formulas, structures, automata and so on. For this we need a *representation* of these objects as strings. Since these objects typically are finite this is not a problem. Once such a representation is fixed an algorithm that operates on these objects can easily be transferred into a Turing machine that decides its corresponding decision problem.
There is one thing to keep in mind about these representations: Their size should typically be polynomially related to the mathematical objects they refer to. Having this in mind, we usually work directly on the mathematical objects instead of on their representations and describe algorithms on these objects instead of Turing machines.

Undecidable Problems. A decision problem is said to be *decidable* if there is a Turing machine that always halts and that accepts exactly the positive instances of this problem and it is *undecidable* if no such Turing machine exists.
Undecidable problems can also be measured in different "degrees" of undecidability. For example a problem might be undecidable but still semi-decidable, i.e. there exists a Turing machine that halts and accepts at least the positive instances of a decision problem but may never halt on negative instances, or it might not even be semi-decidable anymore.
These degrees of undecidability are often classified with respect to a hierarchy formed by first- and second-order arithmetic over the natural numbers, i.e. formulas of first- and second-order logic over the signature with 0, the successor function, addition, multiplication and equality interpreted over the natural numbers. We do not formally distinguish between sets of natural numbers and languages as computed by Turing machines because both models of computation are equivalent and only different representations.
We assume familiarity with the notions of first- and second-order arithmetic for the definitions of these hierarchies, c.f. [26].

We begin with the *arithmetical hierarchy* which classifies sets of natural numbers defined by first-order formulas and fix the above mentioned signature for the remainder of this section.

We say that a relation $R \subseteq \mathbb{N}^k$ for some $k \in \mathbb{N}$ is definable by a first- (or second-)order formula $\varphi(x_1, \dots, x_k)$ with free variables $x_1, \dots, x_k$ if it holds for all tuples $(s_1, \dots, s_k) \in \mathbb{N}^k$ that $(s_1, \dots, s_k) \in R \quad \Leftrightarrow \quad (x_1 \mapsto s_1, \dots, x_k \mapsto s_k) \models \varphi$.

Definition 2.58. A set $S \subseteq \mathbb{N}^k$ for some $k \in \mathbb{N}$ is in Σ^0_0 and Π^0_0 if it is definable by a *quantifier free first-order formula.*

Moreover, it is in Σ^0_{n+1} (resp. Π^0_{n+1}) for some $n \in \mathbb{N}$ if it is definable by a formula $\exists x_1. \dots \exists x_k. \varphi(x_1, \dots, x_k)$ (resp. $\forall x_1. \dots \forall x_k. \varphi(x_1, \dots, x_k)$) for some $k \in \mathbb{N}$ with $\varphi \in \Pi^0_n$ (resp. $\varphi \in \Sigma^0_n$).

As it turns out the decidable sets, i.e. the sets computable by a Turing machine that always halts form the basis of this hierarchy, namely $\Sigma^0_0 = \Pi^0_0$. And the above mentioned semi-decidable problems form the class Σ^0_1.

Further, it is easy to see that the complements of the problems in Σ^0_n form the class Π^0_n. Thus, Π^0_1 encompasses all complements of semi-decidable sets. The connection to Turing machines continues upwards. The class Σ^0_{n+1} can also be characterised as the semi-decidable sets with access to an oracle (a miraculous problem solver that gives a correct answer for the problem in question in one computational step) that decides a problem in Σ^0_n or Π^0_n by Post's Theorem [58].

The *analytical hierarchy* extends the arithmetical hierarchy even further and also accounts for sets definable by second-order arithmetic.

Definition 2.59. A set $S \subseteq \mathbb{N}^k$ for some $k \in \mathbb{N}$ is in Σ^1_0 and Π^1_0 if it is definable by a first-order formula, i.e. if it lies somewhere in the arithmetical hierarchy.

Moreover, it is in Σ^1_{n+1} (resp. Π^1_{n+1}) for some $n \in \mathbb{N}$ if it is definable by a second-order formula $\exists X_1. \dots \exists X_k. \varphi(X_1, \dots, X_k)$ (resp. $\forall X_1. \dots \forall X_k. \varphi(X_1, \dots, X_k)$) for some $k \in \mathbb{N}$ with $\varphi \in \Pi^1_n$ (resp. $\varphi \in \Sigma^1_n$).

All problems that lie somewhere in these hierarchies and are not in $\Sigma^0_0 = \Pi^0_0$ are undecidable. But sets that are higher up in one of these hierarchies are increasingly "less decidable". We say that a problem is *highly undecidable* if it lies somewhere in the analytical hierarchy but not in the arithmetical hierarchy.

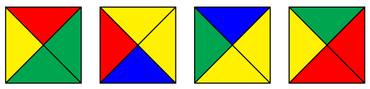
2.11 Tiling Problems

Tiling Problems are a convenient way to show hardness for certain complexity classes.

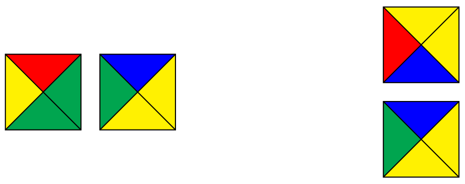
Definition 2.60. A *tiling system* is a tuple $\mathcal{T} = (T, H, V)$ that consists of a set of tiles $T = \{t_1, \ldots, t_m\}$ for some $m \in \mathbb{N}$ together with a horizontal and a vertical matching relation $H, T \subseteq T \times T$.

A tiling problem now consists of a tiling system $\mathcal{T}$ and a region $X \subseteq \mathbb{N} \times \mathbb{N}$ which has to be tiled in such a way that only tiles respecting the horizontal matching relation lie adjacent to each other and only tiles that respect the vertical matching relation lie on top of each other.

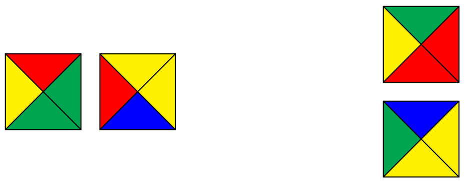
Example 2.61. Consider the tiling system $\mathcal{T} = (T, H, V)$ with the following tiles T:



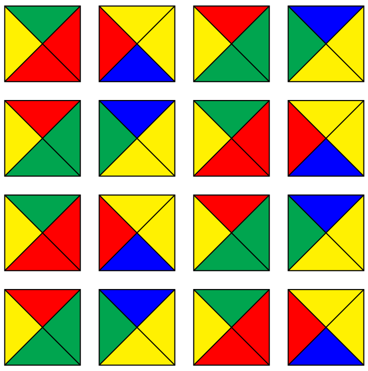
The horizontal and vertical matching relations are given by the colour coding, i.e. only tiles that have matching colors on their respective sides can be placed adjacent or on top of each other. For example the combinations



belong to H resp. V, but the combinations



do not. With these tiles it is possible to successfully tile the 4×4 square:



A closer look at the solution shows that in fact the third and fourth row already repeat the same pattern as the first two rows. Thus, by repeating this pattern it is also possible to tile the whole $4 \times \mathbb{N}$ corridor with this tiling system. And further, we can also repeat this 4×4-pattern starting at the fifth column without violating any of the restrictions so that the whole plane can be tiled.

We will focus mainly on such corridor tiling problems. Let $f : \mathbb{N} \to \mathbb{N}$ be a function. The *corridor of width* $f(n)$ is the subset X of the plane such that $X = \mathbb{N} \times \{0, \ldots, f(n) - 1\}$.

Definition 2.62. Let X be the corridor of width $f(n)$. A *valid* $\mathcal{T}$*-tiling* of the corridor of width $f(n)$ is a function $\tau : X \to T$ such that

- for all $i \in \mathbb{N}$ and all $j \in \{0, \ldots, f(n) - 2\}$ it holds that $(\tau(i, j), \tau(i, j + 1)) \in H$,
- for all $i \in \mathbb{N}$ and all $j \in \{0, \ldots, f(n) - 1\}$ it holds that $(\tau(i, j), \tau(i + 1, j)) \in V$.

Further, there may be initial conditions. We will mainly focus on tiling problems with an initial tile – say t_1 – that is placed at $(0, 0)$.
The $f(n)$*-corridor tiling problem* is now the following:

Input:	a tiling system $\mathcal{T}$ and an $n \in \mathbb{N}$ in unary encoding
Output:	is there a valid $\mathcal{T}$-tiling τ of the corridor of width $f(n)$ with $\tau(0, 0) = t_1$?

The $f(n)$*-corridor* $\mathcal{T}$*-tiling problem* is defined in the same way as the $f(n)$-corridor tiling problem with the exception that the tiling system $\mathcal{T}$ is fixed and the single input is $n \in \mathbb{N}$.
Especially the corridor tiling problems with linear and k-exponential bounded functions are of interest to us.

Proposition 2.63 ([24])**.** For every $k \geq 0$ the 2^n_k-corridor tiling problem is k-ExpSpace-complete.

With 0-fold exponential space we mean polynomial space. Thus the n-corridor tiling problem is PSpace-complete.
Lower bounds can be shown by an encoding of suitable space-restricted Turing machines. Due to the presence of universal Turing machines we can even sharpen these results.

Proposition 2.64. For every $k \geq 0$ there is a tiling system $\mathcal{T}_k$ such that the 2^n_k-corridor $\mathcal{T}_k$-tiling problem is k-ExpSpace-complete.

Corridor tiling problems are particularly useful to prove lower bounds for space-complexity classes. To prove lower bounds for time-complexity classes we can utilise *tiling games* – a two player variant of tiling problems. In this variant one player tries to produce a valid $\mathcal{T}$-tiling of the corridor row by row from bottom to top and from left to right and the other player can make this task harder by choosing the tile placed at the beginning of each row. Of course the other player also has to place his tile in such a way that it does not violate the vertical matching relation.

Definition 2.65. The *$f(n)$-corridor $\mathcal{T}$-tiling game* is played between two players – Adam and Eve – on a corridor of width $f(n)$.
The game is played as follows. The game starts at the bottom left of the corridor at $(0, 0)$ with t_1 already placed.
Whenever the first tile of a row has been placed it is Eve's task to complete this row with tiles that do not violate the horizontal and vertical matching relations given by $\mathcal{T}$.
Whenever a row is finished, Adam can choose to place the first tile of the next row also respecting the vertical matching relation.
A player has won if the other player cannot make a move without violating the horizontal or vertical matching relations. Additionally Eve wins if the game continues forever.

The *$f(n)$-tiling game problem* is now the following:

Input:	a tiling system $\mathcal{T}$ and an $n \in \mathbb{N}$ in unary encoding
Output:	is there a winning strategy for Eve in the $f(n)$-corridor $\mathcal{T}$-tiling game beginning with $\tau(0, 0) = t_1$?

Similarly to corridor-tilings, we define *$f(n)$-corridor $\mathcal{T}$-tiling game* analogously but with regard to a fixed tiling $\mathcal{T}$ and the single input $n \in \mathbb{N}$.
The following results about tiling games are also well-known.

Proposition 2.66 ([24])**.** For every $k \geq 0$ the 2^n_k-corridor tiling game problem is $(k+1)$-ExpTime-hard.

Similarly to corridor-tilings and due to the transformation of deterministic Turing machines into alternating Turing machines that use only logarithmic space compared to the time constraints of its deterministic Turing machine [21] we also obtain the following.

Proposition 2.67. For every $k \geq 0$ there is a tiling system $\mathcal{T}_k$ such that the 2^n_k-corridor $\mathcal{T}_k$-tiling game is $(k+1)$-ExpTime-complete.

Tiling problems do not only help with lower bounds for certain complexity classes. The unbounded problems also help in differentiating decidable and undecidable problems.
The *unbounded corridor tiling problem* is the following:

Input: a tiling system $\mathcal{T}$

Output: is there a valid $\mathcal{T}$-tiling τ of the unbounded corridor, i.e. the corridor with width $\mathbb{N}$, such that $\tau(0,0) = t_1$?

Proposition 2.68 ([91])**.** The *unbounded corridor tiling problem* is undecidable.

However, as noted by Harel [46] this unbounded corridor tiling problem is Co-semi-decidable, i.e. it is in Π^0_1 of the arithmetical hierarchy. In fact, it is Π^0_1-complete.
The *recurring unbounded corridor tiling problem* is another undecidable variant of these tiling problems:

Input: a tiling system $\mathcal{T}$

Output: is there a valid $\mathcal{T}$-tiling τ of the unbounded corridor, i.e. the corridor with width $\mathbb{N}$, such that there are infinitely many $k \in \mathbb{N}$ with $\tau(k,0) = t_1$?

This problem was shown to be highly undecidable.

Proposition 2.69 ([46])**.** The *recurring unbounded corridor tiling problem* is Σ^1_1-complete.

Chapter 3

Hybridisation of Branching-Time Logics

Despite the increase in expressive power from CTL to CTL* these logics still are limited in some ways. For example, they are all invariant under bisimulation which means that there does not exist a formula in any of these logics that can distinguish two bisimilar states. Invariance under bisimulation is the key property for many decision procedures and reasoning methods. It is the reason why all these logics possess the tree-model property, i.e. any satisfiable formula is also satisfied in a tree structure. This simply follows since the tree-unfolding of a structure is bisimilar to the original structure. Moreover, this enables the use of tree-automata and with them an easy test for satisfiability by checking emptiness of an equivalent tree-automaton.

Sometimes however it may be necessary to distinguish bisimilar points. For example if one wants to know if there is a certain state in the system that loops back to itself, possibly meaning that the computation would repeat this cycle over and over and get stuck in this state. Such a property cannot be tested with bisimulation-invariant logics since each structure with a cycle is bisimilar to its tree-unfolding which, by definition, does not have any cycles. However, such "structural" properties can naturally be expressed by first-order logic. For example the formula $\exists x. R(x,x)$ intuitively tests if there is some state x that has a transition back to itself via some accessibility relation R. But this expressive power comes at the cost of the loss of many nice properties. First-order logic is not invariant under bisimulation, does not have a tree-model or even a finite-model property. Moreover, its satisfiability problem is generally undecidable.

Hybrid logics are a framework that aims to combine the good properties of modal and temporal logics with some of the "structural" expressiveness first-order logic. This is done mostly by adding certain "structural" concepts

like *unique names for states* (or existential quantification over states in other words) and referencing these names to various kinds of modal and temporal logics.
Early ideas of adding names for states were already introduced in the 1950s by [78] and the '70s by [19]. However, it was only in the '90s and early 2000s that the ideas were picked up again and gained traction in a series of papers by Goranko [40, 41, 42, 43]. Since then, many hybrid extensions of modal and temporal logics have been studied, see for example [5, 7, 80, 70, 89, 96, 62, 70] to name just a few.
Most of these research articles focus on hybrid extensions of modal logics and basic temporal logics. Hybrid logics that are more expressive – like hybrid extensions of CTL* or the μ-calculus – have generally not yet been studied. The only exceptions are a small comment about the satisfiability problem of hybrid CTL* on trees in [97] and a very restricted "hybrid" extension of the μ-calculus [80].
This chapter is organised as follows. We will first discuss the basic hybrid framework and the ideas behind the hybrid operators considered in this thesis. Then we will present the hybridisations of CTL* and continue to discuss various fragments that coincide with the hybridisation of well-known branching-time logics like CTL, CTL$^+$ or FCTL$^+$. After this, we discuss the difficulties in adding hybrid operators to the μ-calculus and then present the fully hybrid μ-calculus. We finish the chapter with an overview of all obtained hybrid logics and their syntactic relationship to each other before exploring them further in later chapters of this thesis.

3.1 The Main Concepts of Hybrid Logic

Hybrid Logics in their current manifestation feature three types of concepts that are being added to the branching-time logics discussed in Chapter 2.

Names for States. The first concept are *unique names for states* which can be used in combination with the other concepts to identify a single state up to equality. These come in two different varieties: The first one being static names for certain important states. These do not change at all during the investigation. The second variety is of a more dynamic nature: These names or variables can be bound in the context of a formula with a new kind of hybrid operator – called the *binder* (or short $\downarrow$) – that names the current state. Thus, hybrid extensions can name states "on-the-fly".
To make the distinction between static names and dynamically bound variables clearer we will explicitly distinguish these two and will refer to static

names as *nominals* and interpret them as part of an (extended) Kripke structure. Dynamically bound variables will simply be called *state variables* or *variables*.

Identifying Named States. The addition of names – either nominals or variables – by itself does not really add much without a way to actually use this new concept. This is achieved via *tests* that are able identify a named state. These tests are on the surface very similar to simple atomic propositions. For example the formula $x \wedge \mathsf{EX}x$ states that we are at a state called x and there is a transition back to itself. Thus it is equivalent to the first-order formula $E(x, x)$. However, since such names will be *unique* to a referenced state this simple test has a lot of impact.
These simple tests are what breaks the bisimulation-invariance of ordinary branching-time logics. We can distinguish two otherwise bisimilar states simply by naming one of them and testing for it.
From a first-order perspective nominals can simply be seen as "constants" and state variables take the role of simple variables. Similarly, the binder that names a state dynamically in the context of a formula can be seen as a kind of an existential quantification for a new variable with a distinctly modal flavor. For example consider the similarities between the formula $\downarrow x.\mathsf{EX}x$ which intuitively states that we name the current state x and then can transition back to itself and the first-order formula $\exists x.\, E(x, x)$. The first-order formula has a more global nature than the hybrid formula but aside from that both properties are quite similar. From a modal perspective a named state can simply be seen as a state labeled with a special atomic proposition x that happens to always hold on a single state only. And the binder acts as a way to dynamically change these special propositions.

Referencing Names. The concepts of names and tests – while already adding a lot of expressive power to a logic – are usually not that convenient because most modal and temporal logics are by design quite local in their nature which restricts the usage of these concepts a bit. To enhance the use in a more global manner we also add a way to reference these names via a new jump operator – or short $@_x$ – from anywhere in the structure. For example the formula $p \leftrightarrow @_x\, p$ intuitively states that p holds at the current state if and only if p holds at the named state x. It opens up a new and more global way for modal or temporal formulas to transition a structure.

A Plan of Action. We now want to enrich branching-time logics like CTL* and later also the modal μ-calculus with these concepts. Remember that

branching-time logics like CTL* naturally feature *state and path formulas*. Thus, for the two new operators jump and binder we have a choice: Both operators could be allowed as only state formulas or also as path formulas. Consider the binder first. State formulas are evaluated with regard to a state in the structure. So we can easily bind a variable to this current state in the context of a state formula. However, path formulas do also have a current state since they are evaluated with respect to a path and a moment k on this path. Thus, we can simply bind a variable to the k-th state on this path.
Now consider the jump operator. Obviously in the context of a state formula this operator also makes sense. Simply change the current state to the state referenced by the variable of the jump operator. Is it also possible to have the jump operator as part of a path formula? This question is a bit trickier. It turns out that it is possible to allow jump as a path formula – with certain restrictions on how to use jumps. One way to give a well-defined semantics makes sure that a jump in the context of a path evaluation also stays on the same path, i.e. simply jumps back to a previous moment that we have already seen. In this way we can simply continue the path evaluation with the same path but at another moment in time. Such a restriction is realised for example if the variable to which we want to jump is already bound during the context of the same path formula. Thus, we only allow jumps to variables that have been bound on the same path formula.

Remark 3.1. In [97] – which mostly studies hybrid extensions of CTL on trees – there is a minor comment about the complexity of the satisfiability problem of a hybrid version of CTL* where these thoughts about the interaction between binder, jumps and paths were not considered. In fact the reduction they use to show a nonelementary lower bound for the satisfiability problem over trees uses jumps that refer to variables that were placed on another path. As a consequence we believe that this result about the satisfiability problem does not transfer to our hybrid extensions of CTL*.

These considerations lead us to three different kinds of hybridisation for branching-time logics: One in which both hybrid operators are only allowed as state formulas, one in which the binder is allowed as a path formula while the jump is still only allowed as a state formula and one in which both can be used as path formulas (with a small restriction on jumps as mentioned above). These three extensions of branching-time logics will be distinguished by adding different indices to their names: ss, ps and pp indicating that the binder and jump operator (in this order) are allowed only as a state formula – indicated by an s – or also as a path formula – indicated by a p.
The "hybridisation" sp in which the binder is only allowed as a state formula and the jump is allowed as a path formula is not considered because of

the already discussed restrictions needed for jumps to give a well-defined semantics.
In Section 3.5, we will also add these hybrid concepts to the modal μ-calculus. Syntactically, this is much easier for the μ-calculus since it only features one type of formulas. However, adding and mixing these hybrid concepts with least and greatest fixed points on the semantical side is not as straightforward as for the other branching-time logics. We will discuss some of the unforeseen consequences in mixing these concepts and how to obtain a well-behaved semantics later in this chapter.

3.2 Hybridisation of CTL*

Let $\mathit{Var} = \{x, y, \ldots\}$ and $\mathit{Nom} = \{n, m, \ldots\}$ be countable and disjoint sets of first-order variables for states in a Kripke structure. We will refer to the elements of Var as *variables* and to those of Nom as *nominals*. The only difference between Var and Nom is that nominals receive a *fixed* interpretation in a Kripke structure, whereas the interpretation of variables can change dynamically during the evaluation of a formula in a Kripke structure.

Definition 3.2. A *hybrid Kripke structure* is a $\mathcal{K} = \langle S, \to, L\rangle$ like an ordinary Kripke structure except that $L : (\mathit{Prop} \to 2^S) \cup (\mathit{Nom} \to S)$ assigns *sets* of states to each atomic proposition and, at the same time, a *single* state to nominals.

In what follows we only consider hybrid Kripke structures but will usually simply refer to them as Kripke structures.

3.2.1 HCTL$^*_{\mathsf{ss}}$

Formulas of the simplest hybridisation – HCTL$^*_{\mathsf{ss}}$– are given by the grammar

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi \mid n \mid @_n\,\varphi \mid x \mid @_x\,\varphi \mid {\downarrow}x.\varphi \\ \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi,\end{aligned}$$

with $p \in \mathit{Prop}$, $n \in \mathit{Nom}$, $x \in \mathit{Var}$.
To account for the new hybrid operators we extend the interpretation of branching-time logics by a variable assignment $\sigma : \mathit{Var} \to S$ in order to give meaning to free variables in the evaluation of formulas. State formulas are now interpreted with respect to a (hybrid) Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$, a state $s \in S$ and a variable assignment $\sigma : \mathit{Var} \to S$. Non-hybrid state formulas and path formulas are interpreted as before – with the added

presence of the variable assignment. The new hybrid satisfaction relation $\models_{\mathsf{ss}}$ for the state formulas is then given as follows:

$$\begin{array}{lll}
\mathcal{K}, s, \sigma \models_{\mathsf{ss}} n & \text{iff} & L(n) = s, \\
\mathcal{K}, s, \sigma \models_{\mathsf{ss}} @_n \varphi & \text{iff} & \mathcal{K}, L(n), \sigma \models_{\mathsf{ss}} \varphi, \\
\mathcal{K}, s, \sigma \models_{\mathsf{ss}} x & \text{iff} & \sigma(x) = s, \\
\mathcal{K}, s, \sigma \models_{\mathsf{ss}} \downarrow x.\varphi & \text{iff} & \mathcal{K}, s, \sigma[x \mapsto s] \models_{\mathsf{ss}} \varphi, \\
\mathcal{K}, s, \sigma \models_{\mathsf{ss}} @_x \varphi & \text{iff} & \mathcal{K}, \sigma(x), \sigma \models_{\mathsf{ss}} \varphi,
\end{array}$$

where $\sigma[x \mapsto s]$ denotes the update of the variable assignment σ at x, i.e. $\sigma[x \mapsto s](y) = \sigma(y)$ for all variables $y \neq x$ and $\sigma[x \mapsto s](x) = s$.
The operator $\downarrow x.\varphi$ acts as a variable binder; all occurrences of x in φ are *bound* and not free. We assume that $\text{HCTL}^*_{\mathsf{ss}}$ formulas (and all formulas of the ensuing hybrid logics) are *closed*, i.e. do not contain free variables. This is not a restriction at all; a free variable acts like a nominal. Hence, a free variable x in a formula φ can be "removed" by seeing φ as a formula over $\mathit{Var} \setminus \{x\}$ and $\mathit{Nom} \cup \{x\}$, and renaming all other bound occurrences of x in φ. However, in some proofs we might have to deal with free variables that occur during evaluations of subformulas. In these cases we refer to the free variables of a formula φ as $\mathsf{free}(\varphi)$.

Example 3.3. With hybrid and temporal operators combined we can identify structural properties that neither CTL* nor first-order logic by themselves can express. For example, the formula $\downarrow x.\mathsf{EXF}x$ is true in a state s if and only if there is a cycle (of arbitrary but finite length) starting at the state it is evaluated at.

3.2.2 $\text{HCTL}^*_{\mathsf{ps}}$

The second hybridisation – $\text{HCTL}^*_{\mathsf{ps}}$– also allows the binder as a path-formula:

$$\begin{array}{lll}
\varphi & ::= & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi \mid n \mid @_n \varphi \mid x \mid @_x \varphi \mid \downarrow x.\varphi \\
\psi & ::= & \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi \mid \downarrow x.\psi.
\end{array}$$

State formulas are interpreted as in $\text{HCTL}^*_{\mathsf{ss}}$. Path formulas are now interpreted with respect to a Kripke structure $\mathcal{K}$, a path π, a moment k on this path and a variable assignment $\sigma : \mathit{Var} \to S$. Non-hybrid path formulas are also interpreted as before and the new binder is interpreted as follows:

$$\mathcal{K}, \pi, k, \sigma \models_{\mathsf{ps}} \downarrow x.\psi \quad \text{iff} \quad \mathcal{K}, \pi, k, \sigma[x \mapsto \pi^k] \models_{\mathsf{ps}} \psi.$$

We use the suffix ps to indicate that this is the satisfaction relation for $\mathrm{HCTL}^*_{\mathsf{ps}}$.
The interplay between temporal operators on a path formula and the binder brings considerable expressive power. For example it is easy to see that there are formulas that are only satisfied on infinite Kripke structures.

Example 3.4. The formula $\mathsf{EG}{\downarrow}x.\mathsf{XG}\neg x$ expresses that there is a path such that all states along this path are never seen again. Thus, this formula can only be satisfied by an infinite structure because on an infinite path in a finite structure states are bound to be repeated.

We will show later in Section 5.2 that this property cannot be expressed by $\mathrm{HCTL}^*_{\mathsf{ss}}$.

3.2.3 $\mathrm{HCTL}^*_{\mathsf{pp}}$

The final hybridisation – $\mathrm{HCTL}^*_{\mathsf{pp}}$– then also allows jumps as path formulas. The formulas are produced by the grammar

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi \mid n \mid @_n\,\varphi \mid x \mid @_x\,\varphi \mid {\downarrow}x.\varphi\\ \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi \mid {\downarrow}x.\psi \mid @_x\,\psi.\end{aligned}$$

However, as discussed above, "jumping to the state bound to x and then evaluating the path formula ψ" as prescribed by $@_x\,\psi$ only makes sense if x is bound to a moment on that particular path. To ensure that this is the case we restrict the syntax with respect to the interplay between path quantifiers, binders and jumps over path formulas: for any $x \in \mathit{Var}$ and any subformula $@_x\,\psi$ in which ψ is not a state formula we require that there is no occurrence of a path quantifier E (or A) between $@_x\,\psi$ and the smallest ${\downarrow}x.\psi'$ in the syntax tree above $@_x\,\psi$.
To give proper meaning to the jump it does not suffice to just store the state to which we jump because on a path this particular state may occur more than just once. Thus, we additionally store the moment on the path to which the jump operator refers and extend the semantics by a function $\vartheta : \mathit{Var} \to \mathbb{N}$.
The new satisfaction relation $\models_{\mathsf{pp}}$ is then given as follows: State formulas are interpreted as before. Non-hybrid path formulas are also interpreted as before – with the added presence of σ and ϑ. Binder and jump as path formulas are interpreted in the following way:

$$\begin{aligned}\mathcal{K},\pi,k,\sigma,\vartheta \models_{\mathsf{pp}} {\downarrow}x.\psi \quad &\text{iff} \quad \mathcal{K},\pi,k,\sigma[x\to\pi^k],\vartheta[x\to k] \models_{\mathsf{pp}} \psi,\\ \mathcal{K},\pi,k,\sigma,\vartheta \models_{\mathsf{pp}} @_x\,\psi \quad &\text{iff} \quad \mathcal{K},\pi,\vartheta(x),\sigma,\vartheta \models_{\mathsf{pp}} \psi.\end{aligned}$$

Example 3.5. The formula $\mathsf{A}\big(\mathsf{G}\downarrow x.\mathsf{XF}\bigwedge_{i=0}^{n}\bigwedge_{p\in Prop}(\mathsf{X}^i p \leftrightarrow @_x\,\mathsf{X}^i p)\big)$ states that on all paths every sequence of labels that is observed over n steps along the path is repeated somewhere later on this path.

Syntactic conventions for branching-time formulas that were introduced in Section 2.5 can be generalised for hybrid formulas in a straightforward manner.
The notions of subformulas and Fischer-Ladner closure generalise to hybrid formulas with the addition of the clauses $\mathsf{Sub}(x) := \{x\}$ and $\mathsf{Sub}(n) := \{n\}$ for $x \in Var$ and $n \in Nom$. The notion of operators used in Section 2.5 also extends to $\downarrow$ and $@$.
Also, the temporal nesting depth of a hybrid formula is extended by the clauses $\mathsf{nd}(x) = \mathsf{nd}(n) := 0$ and $\mathsf{nd}(\downarrow x.\psi) = \mathsf{nd}(@_x\,\psi) := \mathsf{nd}(\psi)$. Thus, we do not count hybrid operators towards the temporal nesting depth of a formula.
A hybrid branching-time formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ is said to be in *negation normal form* if negation only occurs directly in front of an atomic formula p, x or n.

Lemma 3.6. For each formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ an equivalent formula $\varphi' \in \mathrm{HCTL}^*_{\mathsf{pp}}$ in negation normal form can be computed in linear time.

Proof. Using the equivalences $\neg\downarrow x.\varphi \equiv \downarrow x.\neg\varphi$, $\neg\,@_x\,\varphi \equiv @_x\,\neg\varphi$ as well as de Morgan's laws and the usual temporal equivalences that are used to show Lemma 2.29 we can easily push negations inwards until we obtain a formula in negation normal form. $\square$

3.3 A Syntactical Hierarchy and a Unifying Semantics

Looking at the grammars for $\mathrm{HCTL}^*_{\mathsf{ss}}$, $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}}$ it is quite easy to see that the logics form a syntactical hierarchy with $\mathrm{HCTL}^*_{\mathsf{ss}}$ being the smallest and $\mathrm{HCTL}^*_{\mathsf{pp}}$ being the largest.
A natural question that immediately arises concerns the precise relation of the expressive power of $\mathrm{HCTL}^*_{\mathsf{ss}}$, $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}}$. Note that there exist subtle differences in their semantics. For instance the semantics of $\mathrm{HCTL}^*_{\mathsf{pp}}$ is given with respect to two assignments $\sigma : Var \to S$ and $\vartheta : Var \to \mathbb{N}$ whereas formulas of $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{ss}}$ are only evaluated with respect to one – σ. For this reason we use $\models_{\mathsf{ss}}$, $\models_{\mathsf{ps}}$ and $\models_{\mathsf{pp}}$ to distinguish the satisfaction relations for $\mathrm{HCTL}^*_{\mathsf{ss}}$, $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}}$ respectively. However, it is not too difficult to see that these semantics are conservative in the following sense.

Proposition 3.7. Let $\varphi \in \text{HCTL}^*_{\mathsf{pp}}$ be a closed (state) formula, $\mathcal{K} = \langle S, \rightarrow, L\rangle$ a Kripke structure, $s \in S$, $\sigma_1, \sigma_2 : \mathit{Var} \rightarrow S$ and $\vartheta_1, \vartheta_2 : \mathit{Var} \rightarrow \mathbb{N}$. Then the following statements hold:

a) If $\varphi \in \text{HCTL}^*_{\mathsf{ss}}$ then $\mathcal{K}, s, \sigma_1 \models_{\mathsf{ss}} \varphi$ if and only if $\mathcal{K}, s, \sigma_2 \models_{\mathsf{ss}} \varphi$.

b) If $\varphi \in \text{HCTL}^*_{\mathsf{ps}}$ then $\mathcal{K}, s, \sigma_1 \models_{\mathsf{ps}} \varphi$ if and only if $\mathcal{K}, s, \sigma_2 \models_{\mathsf{ps}} \varphi$.

c) If $\varphi \in \text{HCTL}^*_{\mathsf{pp}}$ then $\mathcal{K}, s, \sigma_1, \vartheta_1 \models_{\mathsf{pp}} \varphi$ if and only if $\mathcal{K}, s, \sigma_2, \vartheta_2 \models_{\mathsf{pp}} \varphi$.

This is easy to see because $\text{HCTL}^*_{\mathsf{pp}}$, $\text{HCTL}^*_{\mathsf{ps}}$ and $\text{HCTL}^*_{\mathsf{ss}}$ formulas are assumed to be closed and thus their evaluation does not depend on the initial variable assignment.
Consequently for closed formulas we will often drop σ and ϑ and only write that $\mathcal{K}, s \models_x \varphi$, $x \in \{\mathsf{ss}, \mathsf{ps}, \mathsf{pp}\}$.

Proposition 3.8. Let $\varphi \in \text{HCTL}^*_{\mathsf{pp}}$ be a closed (state) formula. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure and $s \in S$. Then the following holds:

a) If $\varphi \in \text{HCTL}^*_{\mathsf{ss}}$ then $\mathcal{K}, s \models_{\mathsf{ss}} \varphi$ if and only if $\mathcal{K}, s \models_{\mathsf{ps}} \varphi$.

b) If $\varphi \in \text{HCTL}^*_{\mathsf{ps}}$ then $\mathcal{K}, s \models_{\mathsf{ps}} \varphi$ if and only if $\mathcal{K}, s \models_{\mathsf{pp}} \varphi$.

Proof. The proof of this proposition is also quite simple. We only sketch the key ideas. For the first statement we only need to consider that the semantics for $\text{HCTL}^*_{\mathsf{ps}}$ is simply an extension of $\text{HCTL}^*_{\mathsf{ss}}$ which only misses the case for $\downarrow$ as a path formula. For the second statement, observe that while there is an additional variable interpretation $\vartheta : \mathit{Var} \rightarrow \mathbb{N}$ present in the semantics of $\text{HCTL}^*_{\mathsf{pp}}$, it is never actually used for $\text{HCTL}^*_{\mathsf{ps}}$ formulas – because there the case for jumps is missing from path formulas – and thus ϑ is irrelevant for the satisfaction of an $\text{HCTL}^*_{\mathsf{ps}}$ formula. $\square$

Thus, we will use $\models_{\mathsf{pp}}$ as the only satisfaction relation and simply refer to it as $\models$. With this now unified semantics of these hybrid logics we can directly compare them. A first consequence is that the syntactic hierarchy formed by these logics carries over to a semantic one.

Corollary 3.9. $\text{HCTL}^*_{\mathsf{pp}} \succeq \text{HCTL}^*_{\mathsf{ps}} \succeq \text{HCTL}^*_{\mathsf{ss}}$.

A detailed discussion about their precise expressive power with separation and equality results will be presented in Chapter 4 and 5.

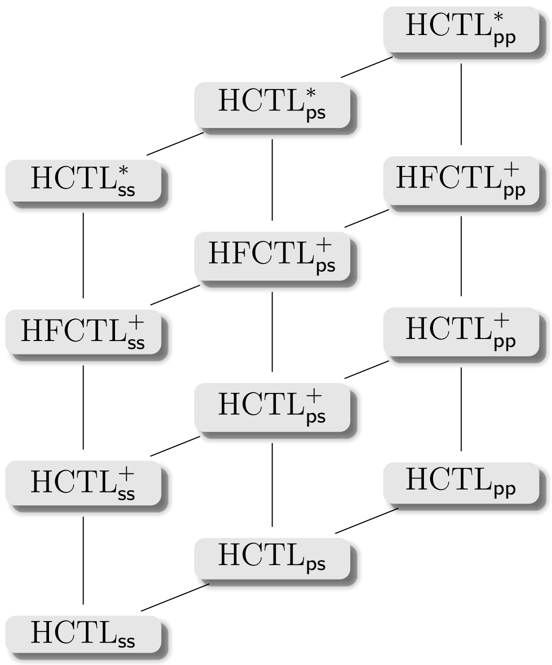

$\mathrm{HCTL}^*_{\mathsf{pp}}$
$\mathrm{HCTL}^*_{\mathsf{ps}}$
$\mathrm{HCTL}^*_{\mathsf{ss}}$
$\mathrm{HFCTL}^+_{\mathsf{pp}}$
$\mathrm{HFCTL}^+_{\mathsf{ps}}$
$\mathrm{HFCTL}^+_{\mathsf{ss}}$
$\mathrm{HCTL}^+_{\mathsf{pp}}$
$\mathrm{HCTL}^+_{\mathsf{ps}}$
$\mathrm{HCTL}^+_{\mathsf{ss}}$
$\mathrm{HCTL}_{\mathsf{pp}}$
$\mathrm{HCTL}_{\mathsf{ps}}$
$\mathrm{HCTL}_{\mathsf{ss}}$

Figure 3.1: Syntactic relationships of all hybrid branching-time logics. The edge relation means that the upper logic extends the lower one.

3.4 Hybridisation of CTL, CTL$^+$ and FCTL$^+$

We will also consider hybridisations of CTL, CTL$^+$ and FCTL$^+$. Initially, as done for CTL* there are three hybridisations available for each logic – ss, ps and pp. They are defined in the same way as the hybridisations for CTL*. Equivalently we can also restrict the non-hybrid path formulas in $\mathrm{HCTL}^*_{\mathsf{ss}}$, $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}}$ to the restrictions shown in Section 2.2. For example the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi \mid n \mid @_n\,\varphi \mid x \mid @_x\,\varphi \mid {\downarrow}x.\varphi$$
$$\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi \mid \mathsf{GF}\varphi \mid {\downarrow}x.\psi \mid @_x\,\psi$$

which disallows nesting of path formulas, allows boolean combinations of them, adds the fairness operator GF and with the same syntactic restriction on path-formulas as for $\mathrm{HCTL}^*_{\mathsf{pp}}$ defines the formulas of $\mathrm{HFCTL}^+_{\mathsf{pp}}$.
Thus, at first we get an additional nine hybridisations based on the logics CTL, CTL$^+$ and FCTL$^+$. These hybridisations and their syntactic relationships are depicted in Figure 3.1.
However, we will now show that each of these syntactic hierarchies based on the hybridisations of CTL, CTL$^+$ and FCTL$^+$ does not carry over to

a semantic one, i.e. $\mathrm{HFCTL}^+_{\mathsf{ss}} \equiv \mathrm{HFCTL}^+_{\mathsf{ps}} \equiv \mathrm{HFCTL}^+_{\mathsf{pp}}$ and the same for the hybrid versions of CTL and CTL$^+$. The idea for this collapse result is quite simple. All we need to do to collapse the hierarchies is to show that binder and jump as path formulas in combination with the restrictions on path formulas below CTL* do not bring anything new to the table in terms of expressive power. This can be seen almost on a syntactic level.
We show this collapse exemplary for $\mathrm{HFCTL}^+_{\mathsf{pp}}$. The same techniques however can also be applied to HCTL$^+$ and HCTL.
We start with the jump operator as a path formula. Inspecting the grammar for path formulas of $\mathrm{HFCTL}^+_{\mathsf{pp}}$ above we see that the jump as a path formula cannot occur under any of the temporal operators since they only allow genuine state formulas underneath them. In addition jumps as part of a path formula can only happen if the variable in question is previously bound on the same path. This means that there cannot be any temporal operator between the binding of the variable and the jump to this variable and consequently only hybrid formulas of type "$\downarrow x.\, @_x\, \psi$" can be built. Thus, we are always on the exact state that we want to jump to which leaves the jump as a path formula meaningless.

Theorem 3.10. For each $\mathrm{HFCTL}^+_{\mathsf{pp}}$ formula φ there is an equivalent formula in $\mathrm{HFCTL}^+_{\mathsf{ps}}$ of size linear in $|\varphi|$.

The case for the binder is also quite simple. Due to the syntactic restrictions that forbid nesting of temporal operators we cannot bind a variable anywhere else than on the first state of a path. However, it does not make any difference if a name is given to the state before or after a path quantifier. Thus, with possible renaming of variables – taking into account that some variables may also occur free in parts of a path formula – we can simply bind the variable right before the path formula begins. The detailed procedure to get rid of the binder is stated in the following lemma and theorem.

Lemma 3.11. Let ψ, ψ' be two $\mathrm{HFCTL}^+_{\mathsf{ps}}$ path formulas such that x has no free occurrences in ψ'. Then the following equivalences are true:

a) $\downarrow x.\downarrow y.\psi(x,y) \equiv \downarrow x.\psi[x/y]$,

b) $\downarrow x.\psi \vee \psi' \equiv \downarrow x.(\psi \vee \psi')$ and

c) $\downarrow x.\psi \wedge \psi' \equiv \downarrow x.(\psi \wedge \psi')$.

Additionally, if x appears free in ψ' and x' is a new variable that does not appear free in ψ or ψ' then the following equivalences are true:

d) $\downarrow x.\psi \vee \psi' \equiv \downarrow x'.(\psi[x'/x] \vee \psi')$,

e) $\downarrow x.\psi \land \psi' \equiv \downarrow x'.(\psi[x'/x] \land \psi')$

Theorem 3.12. For each $\mathrm{HFCTL}^+_{\mathsf{ps}}$ formula with k variables there is an equivalent $\mathrm{HFCTL}^+_{\mathsf{ss}}$ formula with at most $2k$ variables.

Proof. Let $\varphi \in \mathrm{HFCTL}^+_{\mathsf{ps}}$ such that φ uses at most k variables, say $x_1, \ldots, x_k$. We will describe a procedure to translate φ into an $\mathrm{HFCTL}^+_{\mathsf{ss}}$ formula by eliminating binder as path formulas. So, w.l.o.g. assume that $\varphi = \mathsf{E}\psi$ with possible binders as part of the path formula ψ. Furthermore, assume that negations in ψ have already been pushed downwards to the level of maximal state-subformulas with the usual equivalences and the equivalence $\neg \downarrow x.\psi \equiv \downarrow x.\neg\psi$.
The procedure will at most double the number of used variables. So, let $x'_1, \ldots, x'_k$ be fresh variables not used in φ. They will serve as possible replacements for the non-primed variables.
Note that due to the restrictions on nesting path formulas, the binder can only appear on the top level of the syntax-tree of ψ or underneath boolean connectives. We use the equivalences in Lemma 3.11 to push all variables upwards over the boolean connectives. Note that by renaming variables we do not introduce free occurrences of the primed variables. Thus, we never need to rename the primed variables so that we need at most one replacement for each variable.
After pushing all variables upwards to the top-level we use the equivalence $\mathsf{E}\downarrow x.\psi' \equiv \downarrow x.\mathsf{E}\psi'$ to remove variables completely from the path formula, thus obtaining an $\mathrm{HFCTL}^+_{\mathsf{ss}}$ formula. $\square$

Remark 3.13. The translation from $\mathrm{HFCTL}^+_{\mathsf{ps}}$ to $\mathrm{HFCTL}^+_{\mathsf{ss}}$ is only linear in the *length* of the formula. Due to the renaming of variables the blowup in *size* can be exponential. It is still an open question if there is a translation that involves only a polynomial blowup.

For HCTL the translation is even simpler, since there are not even boolean combinations allowed underneath a path quantifier. Thus, if a variable is bound directly underneath a path quantifier we can just push it over the next path quantifier and achieve a linear translation from $\mathrm{HCTL}_{\mathsf{pp}}$ to $\mathrm{HCTL}_{\mathsf{ss}}$ in the size of the formula.

3.5 The Fully Hybrid μ-calculus

Adding hybrid operators to the μ-calculus is – at least syntactically – a lot more straightforward than for the other branching-time logics. We only have one type of formula, so we can simply add the three hybrid operators to the

μ-calculus. Thus, syntactically the *Fully Hybrid μ-Calculus H_μ* is given by the grammar

$$\varphi := p \mid n \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid @_n\,\varphi \mid {\downarrow} x.\varphi \mid \mu X.\varphi(X)$$

where $p \in \mathit{Prop}$, $n \in \mathit{Var} \cup \mathit{Nom}$, $x \in \mathit{Var}$ and $X \in \mathit{Var}_2$ and the same syntactic restrictions for fixed points as in the modal μ-calculus apply to make sure that the induced operators are still monotone.
However, as we will now see the semantics become quite a bit more involved when first- and second-order variables are mixed together with least and greatest fixed points.

3.5.1 A first try at a proper semantics

Sattler and Vardi [80] already considered a hybrid form of the μ-calculus called the *hybrid μ-calculus*. However, their version only featured nominals and jumps and no dynamic naming of states in form of the binder. To properly distinguish both logics we have named our logic the *fully hybrid μ-calculus*.
A naïve attempt to give a proper semantics to the fully hybrid μ-calculus would be to take theirs and simply enrich it with a variable assignment for the first-order variables as done in the previous sections for the other branching-time logics.
Let $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure, $\rho : \mathit{Var}_2 \to 2^S$ be an assignment for the second-order variables, $\sigma : \mathit{Var} \to S$ an assignment for the first-order variables and $\varphi \in \mathrm{H}_\mu$. We interpret formulas now with respect to $\mathcal{K}, \rho$ and σ. Following [80], we get for nominals $n \in \mathit{Nom}$, that

$$\begin{aligned} [\![n]\!]^{\mathcal{K}}_{\rho,\sigma} &= L(n) \text{ and} \\ [\![@_n\,\varphi]\!]^{\mathcal{K}}_{\rho,\sigma} &= \begin{cases} S & \text{if } L(n) \in [\![\varphi]\!]^{\mathcal{K}}_{\rho,\sigma}, \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

A nominal passes its test at the state at which the nominal holds and $@_n\,\varphi$ is satisfied everywhere if φ is satisfied at n.
For variable tests and jumps on variables $x \in \mathit{Var}$ we could then simply do the same but with reference to the assignment σ of the first-order variables and the binder simply updates the variable assignment:

$$\begin{aligned} [\![x]\!]^{\mathcal{K}}_{\rho,\sigma} &= \sigma(x), \\ [\![@_x\,\varphi]\!]^{\mathcal{K}}_{\rho,\sigma} &= \begin{cases} S & \text{if } \sigma(x) \in [\![\varphi]\!]^{\mathcal{K}}_{\rho,\sigma}, \\ \emptyset & \text{otherwise,} \end{cases} \end{aligned}$$

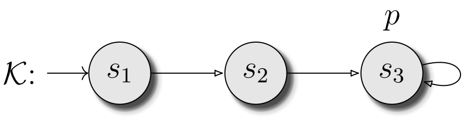
$$[\![\downarrow x.\varphi]\!]^{\mathcal{K}}_{\rho,\sigma} = \{s \in S \mid s \in [\![\varphi]\!]^{\mathcal{K}}_{\rho,\sigma[x \to s]}\}.$$

The semantics is well-defined. One can easily check that the presence of jumps and binders does not change the monotonicity of the induced function, i.e. the functions $V \mapsto [\![\downarrow x.\varphi(X)]\!]^{\mathcal{K}}_{\rho[X \mapsto V],\sigma}$ and $V \mapsto [\![@_x\, \varphi(X)]\!]^{\mathcal{K}}_{\rho[X \mapsto V],\sigma}$ are still monotone if all free occurrences of X are under an even number of negations. Thus, with Theorem 2.20 least and greatest fixed points over the power-set lattice still exist.
However, we will now argue that this semantics is *not convenient* because there are some unforeseen and undesirable implications.

Example 3.14. Consider for example the formula

$$\mu Y.(p \wedge \neg x) \vee \downarrow x.\Diamond Y.$$

To illustrate why this first attempt at a proper semantics is not useful we will evaluate this fixed point on the Kripke structure



and take a look at the approximations of this fixed point.
We evaluate both the syntactic unfoldings as well as the actual approximations for the variable assignment $\sigma(x) = s_3$ under the proposed semantics. We begin with the actual approximations for the fixed points as defined in Section 2.4.4: We have $\mu Y^0.(p \wedge \neg x) \vee \downarrow x.\Diamond Y = \emptyset$ by definition and

$$\mu Y^1.(p \wedge \neg x) \vee \downarrow x.\Diamond Y = [\![\mu Y.(p \wedge \neg x) \vee \downarrow x.\Diamond Y]\!]^{\mathcal{K}}_{\{X \mapsto \emptyset, x \mapsto s_3\}} = \emptyset$$

since there is no state that satisfies $(p \wedge \neg x)$ or that has a successor in the emptyset. Thus the empty set is the least fixed point of this formula under the proposed semantics.
We now evaluate the syntactic unfoldings of this fixed point. To start we have

$$[\![(p \wedge \neg x) \vee \downarrow x.\Diamond \mathtt{ff}]\!]^{\mathcal{K}}_{x \mapsto s_3} = \emptyset$$

which evaluates to the same as the first actual approximation. However, the second syntactic unfolding evaluates to

$$[\![(p \wedge \neg x) \vee \downarrow x.\Diamond\big((p \wedge \neg x) \vee \downarrow x.\Diamond \mathtt{ff}\big)]\!]^{\mathcal{K}}_{x \mapsto s_3} = \{s_2\}.$$

The reason for this is that at first – when x is bound to s_3 – there is no state which satisfies $(p \wedge \neg x)$. However, at s_2 we can rebind the variable x to s_2

and then find a successor (s_3) that now satisfies $(p \wedge \neg x)$. To continue we get for the third syntactic unfolding:

$$[\![(p \wedge \neg x) \vee \downarrow x.\Diamond\big((p \wedge \neg x) \vee \downarrow x.\Diamond\big((p \wedge \neg x) \vee \downarrow x.\Diamond \mathtt{ff}\big)\big)]\!]^{\mathcal{K}}_{x \mapsto s_3} = \{s_2, s_1\}.$$

This also stabilises here but obviously differs from the least fixed point calculated above.

Thus, we have that under this semantics in general the equivalence

$$\mu Y.\varphi(Y) \equiv \varphi(\mu Y.\varphi(Y))$$

does not hold anymore and with this it also follows that in general

$$[\![\varphi^{\alpha}(X)]\!]^{\mathcal{K}}_{\rho,\sigma} \neq \mu X^{\alpha}.\varphi(X),$$

meaning that syntactical unfoldings can in general not be used anymore to approximate fixed points.
To find the reason why the syntactic unfolding principle does not work anymore under these semantics, consider again the first syntactic unfolding $(p \wedge \neg x) \vee \downarrow x.\Diamond((p \wedge \neg x) \vee \downarrow x.\Diamond \mathtt{ff})$. The second $(p \wedge \neg x)$ gets evaluated with regard to an updated first-order assignment caused by the outermost binder $\downarrow x$.
Thus, when evaluating the formula at s_2 the second disjunct is satisfied because we can bind x to s_2 with the first binder and then evaluate $\Diamond((p \wedge \neg x) \vee \downarrow x.\Diamond \mathtt{ff})$ which of course evaluates to true because if x is bound to s_2 we can transition to s_3 which then satisfies $(p \wedge \neg x)$.
On the other hand, under the proposed semantics the actual approximations are computed with regard to a fixed variable assignment which never changes – here $\sigma(x) = s_3$. Thus, we essentially get that the formula $\mu Y.(p \wedge \neg x) \vee \downarrow x.\Diamond Y$ is equivalent to the formula $\mu Y.(p \wedge \neg x) \vee \Diamond Y$ since the binder is ignored completely in the fixed point calculation.
To make things even worse one can quickly check that for the other variable assignments $\sigma(x) = s_1$ and $\sigma(x) = s_2$ in the example above, the approximations and syntactical unfoldings do indeed match and calculate the same fixed point.
Hence, the fully hybrid μ-calculus with the currently proposed semantics would be a logic with fixed point calculations that may or may not differ from their *intended* syntactical unfoldings depending on the variable assignment and in some cases binding a variable would not even have any effect on the formula. Thus it would be even harder to properly understand a fixed point formula.

We feel however that the syntactical unfoldings of such a fixed point formula accurately matches the intention of writing a hybrid fixed point formula and thus should be reproduced by the semantics. To solve this problem we propose a different semantics for H_μ in the next section. This semantics will also build upon the hybrid μ-calculus proposed by Sattler and Vardi in [80] but also preserves the familiar features and traits known from L_μ.

3.5.2 Hybridisation of the μ-calculus

To solve the inconsistency with the previous semantics we integrate the variable assignment even more and place it on the same level as a state.
The meaning of a formula thus is a set of pairs that consist of a state s and a variable assignment σ for all state variables. Fixed points will be calculated on the powerset-lattice over the cartesian product of states and mappings of state variables and consequently the interpretation of free second-order variables will also be lifted to the type $\rho : \mathit{Var}_2 \to 2^{S\times(\mathit{Var}\to S)}$.
Formally the semantics of a formula $\varphi \in \mathrm{H}_\mu$ is given with respect to a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ and an assignment $\rho : \mathit{Var}_2 \to 2^{S\times(\mathit{Var}\to S)}$ with $[\![\varphi]\!]^{\mathcal{K}}_\rho \subseteq S \times (\mathit{Var} \to S)$:

$$\begin{aligned}
[\![p]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid s \in L(p)\},\\
[\![n]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid s = L(n)\},\\
[\![x]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid s = \sigma(x)\},\\
[\![X]\!]^{\mathcal{K}}_\rho &= \rho(X),\\
[\![\neg\varphi]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid (s,\sigma) \notin [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}}_\rho &= [\![\varphi_1]\!]^{\mathcal{K}}_\rho \cup [\![\varphi_2]\!]^{\mathcal{K}}_\rho,\\
[\![\Box\varphi]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid \forall t \in S:\ \text{if } s \to t,\ \text{then } (t,\sigma) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![@_x\,\varphi]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid (\sigma(x),\sigma) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![@_n\,\varphi]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid (L(n),\sigma) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![\downarrow x.\varphi]\!]^{\mathcal{K}}_\rho &= \{(s,\sigma) \mid (s,\sigma[x \mapsto s]) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![\mu X.\varphi(X)]\!]^{\mathcal{K}}_\rho &= \bigcap\{T \subseteq S \times (\mathit{Var} \to S) \mid [\![\varphi]\!]^{\mathcal{K}}_{\rho[X\to T]} \subseteq T\}
\end{aligned}$$

with $p \in \mathit{Prop}$ $n \in \mathit{Nom}$, $x \in \mathit{Var}$ and $X \in \mathit{Var}_2$. We write $\mathcal{K}, s, \sigma, \rho \models \varphi$ if $(s,\sigma) \in [\![p]\!]^{\mathcal{K}}_\rho$. For closed formulas we also may drop σ and ρ.
Note that for formulas with no state variables we get the same semantics as the hybrid μ-calculus defined by Sattler and Vardi [80]. Moreover, for formulas with no state variables and no nominals we simply obtain the usual semantics of the modal μ-calculus L_μ.

With this semantics we can again, as for L_μ show the following result.

Theorem 3.15. Let $\varphi(X)$ be a H_μ formula with free variable X, $\psi \in H_\mu$ a closed formula, $\mathcal{K}$ be any Kripke structure and ρ a variable assignment as before. Then

$$[\![\varphi[\psi/X]]\!]^{\mathcal{K}}_{\rho} = [\![\varphi(X)]\!]^{\mathcal{K}}_{\rho[X \mapsto V]}$$

where $V = [\![\psi]\!]^{\mathcal{K}}_{\rho}$.

Proof. We can prove this by a straightforward induction on φ. The only interesting case is the binder. So, let $\varphi = \downarrow x.\varphi_1(X)$. Then

$$\begin{aligned}
[\![\varphi[\psi/X]]\!]^{\mathcal{K}}_{\rho} &= [\![\downarrow x.\varphi_1[\psi/X]]\!]^{\mathcal{K}}_{\rho} \\
&= \{(s,\sigma) \in S \times (\mathit{Var} \to S) \mid (s, \sigma[x \mapsto s]) \in [\![\varphi_1[\psi/X]]\!]^{\mathcal{K}}_{\rho}\} \\
&\overset{\text{IH}}{=} \{(s,\sigma) \in S \times (\mathit{Var} \to S) \mid (s, \sigma[x \mapsto s]) \in [\![\varphi_1(X)]\!]^{\mathcal{K}}_{\rho[X \mapsto V]}\} \\
&= [\![\downarrow x.\varphi_1(X)]\!]^{\mathcal{K}}_{\rho[X \mapsto V]},
\end{aligned}$$

where the first equality is simply the definition of φ and the second and fourth equality are by the semantics of the binder. The third equality uses the induction hypothesis for φ_1. $\square$

Corollary 3.16. For all formulas $\varphi \in H_\mu$ and all finite ordinals α it holds that $[\![\varphi^\alpha(X)]\!]^{\mathcal{K}}_{\rho} = \mu X^\alpha.\varphi(X)$.

In this way we get a well-defined semantics for H_μ that also preserves many of the basic properties of L_μ.

Example 3.17. We take another look at the formula $\mu Y.(p \wedge \neg x) \vee \downarrow x.\Diamond Y$ and the Kripke structure $\mathcal{K}$ from Example 3.14 and re-evaluate it under the newly proposed semantics.
For the first approximation we get that

$$\mu Y^0.(p \wedge \neg x) \vee \downarrow x.\Diamond Y = \{(s_3, x \mapsto s_1), (s_3, x \mapsto s_2)\}$$

because no state has a successor in the empty set and, as before, only if the variable assignment for x does not point to s_3, s_3 satisfies $(p \wedge \neg x)$. For the second approximand we then obtain

$$\begin{aligned}
\mu Y^1.(p \wedge \neg x) \vee \downarrow x.\Diamond Y =& \mu Y^0.(p \wedge \neg x) \vee \downarrow x.\Diamond Y \\
& \cup \{(s_2, x \mapsto s_1), (s_2, x \mapsto s_2), (s_2, x \mapsto s_3)\}
\end{aligned}$$

since we can also start at s_2 bind the variable to s_2 and then find a successor – $(s_3, x \mapsto s_2)$ – in $\rho(Y)$. And lastly this is extended for the third approximand to

$$\begin{aligned}\mu Y^2.(p \wedge \neg x) \vee {\downarrow} x.\Diamond Y =& \mu Y^1.(p \wedge \neg x) \vee {\downarrow} x.\Diamond Y \\ & \cup \{(s_1, x \mapsto s_1), (s_1, x \mapsto s_2), (s_1, x \mapsto s_3)\}\end{aligned}$$

with an analogous reasoning.
If we only focus on the fixed point relative to the initial variable assignment $\sigma(x) = s_3$ we obtain the same fixed point that was calculated for the syntactic unfolding in Example 3.14.

Similarly to the case of $\mathrm{HCTL}^*_{\mathsf{pp}}$ the notions of subformulas, size, length and negation normal form extend to H_μ. Moreover, similar to Lemma 3.6 we can also prove the following for H_μ.

Lemma 3.18. For each formula $\varphi \in \mathrm{H}_\mu$ an equivalent formula $\varphi' \in \mathrm{H}_\mu$ in negation normal form can be computed in linear time.

3.5.3 Model Checking Games for H_μ

Similar to L_μ, formulas in the fully hybrid μ-calculus are notoriously hard to understand. Nested fixed points – now with added hybrid operators to keep in mind – can make it very hard to understand even simple properties. For this reason we also introduce model checking games for H_μ that build on the foundation of model checking games for L_μ. These games provide a framework to reason about satisfaction and dissatisfaction of H_μ formulas and thus also enable us to better understand H_μ formulas.

Definition 3.19. Let $\varphi \in \mathrm{H}_\mu$ be in negation normal form, i.e. negation only occurs directly in front of atomic propositions, variables or nominals and let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure.
The model checking game $\mathcal{G}(\mathcal{K}, \varphi)$ is played between the two players $\boldsymbol{V}$ and $\boldsymbol{R}$. The game is played on the configuration space $S \times (\mathit{Var} \rightarrow S) \times \mathsf{Sub}(\varphi)$. We usually write such a position as $s, \sigma \vdash \psi$.
Intuitively, $\boldsymbol{V}$ tries to show that $\mathcal{K}, s, \sigma \models \varphi$ holds in a position $s, \sigma \vdash \varphi$ while $\boldsymbol{R}$ tries to refute that.
The game begins at some position $s, \sigma \vdash \varphi$ and continues using the rules in Figure 3.2.
The game rules that are annotated by either $\boldsymbol{V}$ or $\boldsymbol{R}$ induce a choice for this player. For example in a configuration $s, \sigma \vdash \Diamond\psi$, $\boldsymbol{V}$ can choose on which successor state t with $s \rightarrow t$ she wants to prove that ψ holds. The game then continues in the configuration $t, \sigma \vdash \psi$.

$$\frac{s,\sigma \vdash \psi_1 \wedge \psi_2}{s,\sigma \vdash \psi_1 \quad s,\sigma \vdash \psi_2}\ (\boldsymbol{R}) \qquad \frac{s,\sigma \vdash @_x\,\varphi}{\sigma(x),\sigma \vdash \varphi} \qquad (\boldsymbol{V})\ \frac{s,\sigma \vdash \psi_1 \vee \psi_2}{s,\sigma \vdash \psi_1 \quad s,\sigma \vdash \psi_2}$$

$$\frac{s,\sigma \vdash \Box\psi}{t,\sigma \vdash \psi}\ (\boldsymbol{R} : s \to t) \qquad \frac{s,\sigma \vdash {\downarrow} x.\varphi}{s,\sigma[x \mapsto s] \vdash \varphi} \qquad (\boldsymbol{V} : s \to t)\ \frac{s,\sigma \vdash \Diamond\psi}{t,\sigma \vdash \psi}$$

$$\frac{s,\sigma \vdash \nu X.\psi(X)}{s,\sigma \vdash \psi(X)}\ (\boldsymbol{R}) \qquad \frac{s,\sigma \vdash X}{s,\sigma \vdash \mathsf{fp}_\varphi(X)} \qquad (\boldsymbol{V})\ \frac{s,\sigma \vdash \mu X.\psi(X)}{s,\sigma \vdash \psi(X)}$$

Figure 3.2: The game rules for the H_μ model checking games.

The winning conditions can mostly be transferred from the model checking games for L_μ as introduced in Section 2.9.3. In addition, configurations of the form $s, \sigma \vdash x$ belong to player $\boldsymbol{R}$ if $s = \sigma(x)$ and to $\boldsymbol{V}$ otherwise.

Similarly to the case of L_μ model checking games these games truly characterise the model checking problem for H_μ.

Theorem 3.20. $\boldsymbol{V}$ has a winning strategy in the model checking game $\mathcal{G}(\mathcal{K}, \varphi)$ starting at position $s, \sigma \vdash \varphi$ if and only if $\mathcal{K}, s, \sigma \models \varphi$.

We will prove this in Subsection 6.2.4 as a side product of the model checking algorithm for H_μ.

3.6 The Hybrid Branching-Time Landscape

In the previous chapter we have introduced three ways to hybridise classical branching-time logics ranging from CTL to CTL*. Together with the hybridisation of the modal μ-calculus this leads to thirteen different hybrid logics. Twelve are based on the logics CTL, CTL$^+$, FCTL$^+$ and CTL* and their syntactic relationships have already been depicted in Figure 3.1.
Generally these three hybridisations form a syntactical hierarchy on top of the branching-time logics to which they are added. Also, each hybridisation is subsumed by the same hybridisation on top of a richer branching-time logic. For example $\mathrm{HCTL}_{\mathsf{ps}}$ is subsumed by $\mathrm{HCTL}^+_{\mathsf{ps}}$ etc.
Figure 3.3 takes into account that by Theorem 3.10 and 3.12 the syntactic hierarchies below $\mathrm{HCTL}^*_{\mathsf{ss}}$ do not carry over to semantic ones, i.e. there is no increase in expressive power. It gives a first overview of the defined hybrid branching-time logics in terms of their expressive power. These relationships are explored in more detail in Chapter 5. Because of the collapse of the hierarchies below $\mathrm{HCTL}^*_{\mathsf{ss}}$, we will mainly focus on the logics depicted in boldface.

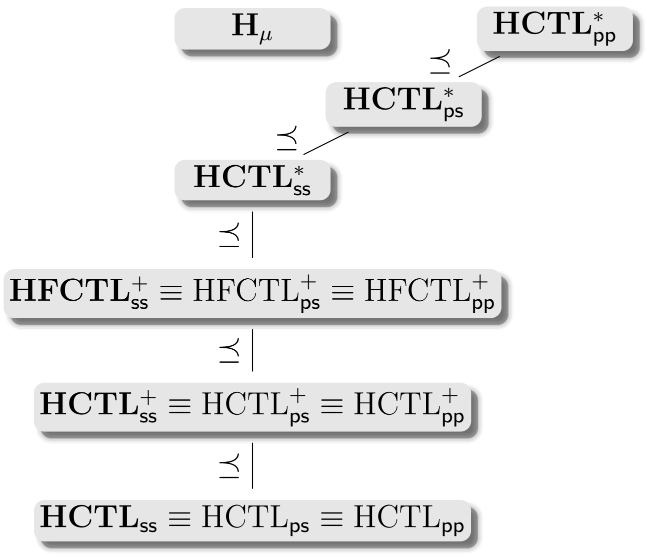

$\mathbf{H}_\mu$
$\mathbf{HCTL}^*_{\mathsf{pp}}$
$\preceq$
$\mathbf{HCTL}^*_{\mathsf{ps}}$
$\preceq$
$\mathbf{HCTL}^*_{\mathsf{ss}}$
$\preceq$
$\mathbf{HFCTL}^+_{\mathsf{ss}} \equiv \mathrm{HFCTL}^+_{\mathsf{ps}} \equiv \mathrm{HFCTL}^+_{\mathsf{pp}}$
$\preceq$
$\mathbf{HCTL}^+_{\mathsf{ss}} \equiv \mathrm{HCTL}^+_{\mathsf{ps}} \equiv \mathrm{HCTL}^+_{\mathsf{pp}}$
$\preceq$
$\mathbf{HCTL}_{\mathsf{ss}} \equiv \mathrm{HCTL}_{\mathsf{ps}} \equiv \mathrm{HCTL}_{\mathsf{pp}}$

Figure 3.3: The hybrid branching-time landscape. A first comparison in terms of expressive power between the introduced hybrid branching-time logics.

Simply by definition the fully hybrid μ-calculus does not have a clear syntactic or even semantic connection to the other hybrid logics. Although since it is based on the very expressive modal μ-calculus which subsumes even CTL* it stands to reason that it may even be more expressive than many of the other hybrid logics or at least somewhere high up in this hierarchy. The relationship of H_μ and the other hybrid logics will be discussed in more detail in Chapter 4 and 5. There we will show that H_μ subsumes $\mathrm{HCTL}^*_{\mathsf{ss}}$ but is incomparable to $\mathrm{HCTL}^*_{\mathsf{ps}}$.

Fragments of Hybrid Logics. The complexity, expressive power and in some cases even the decidability of these hybrid logics often seems closely tied to the number of variables that are used.
For example with n variables it is easy to write a formula that is satisfied if a state has exactly $(n-1)$ successors: We can simply mark the state plus $(n-1)$ of its successors and require that all successor states are marked by one of the last $(n-1)$ variables using $\downarrow$, @ and EX to jump back and forth between the original state and its successors.
However, it is not easy to see if this can also be done with only $(n-1)$ variables since then for every variable assignment at least one successor state is unnamed and can intuitively only be identified up to bisimulation – which may allow duplicates of said state that are bisimilar.

To get a more refined analysis we sometimes use *bounded variable fragments* or simply bounded fragments which are the fragments of these hybrid logics that use at most a certain number of variables. We denote these fragments with an upper index that indicates the maximum number of allowed variables.

Definition 3.21. The *bounded fragments* $\mathrm{H}^k\mathrm{CTL}$, $\mathrm{H}^k\mathrm{CTL}^+$, $\mathrm{H}^k\mathrm{FCTL}^+$, $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ss}}$, $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ps}}$, $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{pp}}$ and H^k_μ for any $k \in \mathbb{N}$ denote the fragments of the indicated logics that use at most k variables.

Also, we may consider fragments that feature only some of the hybrid operators. We will indicate this by adding the *allowed hybrid operators* in parantheses. For example $\mathrm{H}_\mu(\downarrow, x)$ indicates the fragment of the fully hybrid μ-calculus that only features $\downarrow$ and variable tests but no @.
Of course both fragments can also be combined. For example $\mathrm{H}^k_\mu(\downarrow, @)$ refers to the fragment of H^k_μ that disallows variable tests, i.e. only $\downarrow$ and @ are allowed as hybrid operators.

3.7 Hybrid and other extensions of Branching-Time Logics

As a first showcase of what hybrid logics are capable of we compare them with some other recent extensions of branching-time logics. We focus mainly on extensions of CTL^* and its relatives. A comparison of the fully hybrid μ-calculus with an extension of L_μ is shown as part of Chapter 4.
This section is only meant to showcase the hybrid branching-time logics and to get a better grasp of their expressive power. In particular, it is not a rigorous comparison with the other presented extensions of CTL^* and its fragments. Especially smaller details or minor differences either in their semantics or on a syntactic level will not be highlighted. In particular we also do not aim to obtain any complexity bounds (upper or lower) via these connections here. Instead, we will focus more on the similarities to our hybrid branching-time logics. Nevertheless, we will at least mention some possibilities to transfer complexity bounds.

Memoryful CTL^*. Memoryful CTL^*– or short $m\mathrm{CTL}^*$ – introduced in [59] by Kupferman and Vardi is an extension of regular CTL^* interpreted on computation trees. It redefines path quantification to full paths, i.e. paths starting at the root of the tree but the state in which the actual path quantification takes place is remembered as a special state called $\mathsf{present}$.

The motivation for this logic originates in the planning over nondeterministic domains and the desire to have a middle ground between properties that necessarily need to hold on all possible paths and properties that hold on a single path as specified by the path quantifiers in CTL*. Redefining the path quantification as done above thus also allows to make statements about all paths (starting at the root) that go through the current state.
Formally, given a tree $\mathcal{T} = \langle T, \to, L\rangle$ and two states $s, c \in T$, the satisfaction relation for the path quantification in mCTL* is defined via

$$\mathcal{T}, s, c \models \mathsf{E}\psi \text{ iff} \quad \text{there is a path } \pi \text{ starting at } \varepsilon \text{ and going through } s \text{ such that } \mathcal{T}, \pi, 0, s \models \psi.$$

Note that the path evaluation starts at ε and s is only remembered as the special state $\mathsf{present}$:

$$\mathcal{T}, s, c \models \mathsf{present} \quad \text{iff} \quad s = c.$$

This reinterpretation of the path quantification makes it easy to also talk about past events without necessarily adding explicit past operators.

Example 3.22. The formula $\mathsf{AG}(\mathsf{grant} \to \mathsf{EF}(\mathsf{req} \wedge \mathsf{Fpresent}))$ states for example that each grant is preceeded by a request.

The concept of remembering a state is inherently added by hybrid logics via variables and nominals and thus we can use the hybrid framework to express mCTL* formulas. For this, let $\mathcal{T}_{\mathsf{root}}$ be the extension of a tree $\mathcal{T}$ with a single nominal root located at ε.

Theorem 3.23. For each mCTL* formula φ there is an HCTL$^*_{\mathsf{ss}}$ formula $\widehat{\varphi}$ of size linear in φ with only one variable such that $\mathcal{T}, s \models \varphi$ if and only if $\mathcal{T}_{\mathsf{root}}, s \models \widehat{\varphi}$.

Proof. The proof is via a simple translation. We use the hybrid operators to simulate the semantic differences in the path quantification.
An equivalent HCTL$^*_{\mathsf{ss}}$ formula can be obtained by replacing each subformula of type $\mathsf{E}\psi$ formula with $\downarrow \mathsf{present}.\, @_{\mathsf{root}}\, \mathsf{E}(\mathsf{Fpresent} \wedge \psi)$. This hybrid construction binds the variable $\mathsf{present}$ to the current state, jumps back to the root and then defines a path starting at ε that goes through $\mathsf{present}$ and satisfies ψ.
It is clear that this translation is linear in φ. Thus, the proposed statement follows. $\square$

The added nominal at the root of the tree can also be omitted if the formulas are only evaluated at the root of the tree by simply placing a variable root right at the beginning.
mCTL* was shown to be only as expressive as CTL* [59]. Thus, the embedding into HCTL$^*_{\mathsf{ss}}$ also follows via a translation to CTL* which is already syntactic fragment of HCTL$^*_{\mathsf{ss}}$. However, this translation is nonelementary and thus our direct translation is much more preferable.

Cycle-CTL*. Another recent extension of CTL* features explicit reasoning about cycles in a Kripke structure – not only trees anymore. Cycle-CTL* [36] extends CTL* by adding two new path quantifiers $\mathsf{E}^{\circlearrowleft}$ and $\mathsf{A}^{\circlearrowleft}$. The operators can be understood as "there is a *cyclic* path" and "on all *cyclic* paths" where a cyclic path is an infinite path that loops back to its initial state infinitely often.
Many decision procedures on graph-based tools such as automata reduce to detecting specific cyclic behaviour. The most prominent ones are possibly non-emptiness checks for Büchi-automata which are essentially a simple reachability check for an accepting state and a search for a cycle from this accepting state. Other examples are mentioned in [36] as well. For this reason a logic to specify such properties may indeed be useful.
Again, it almost seems obvious that Cycle-CTL$^* \preceq$ HCTL$^*_{\mathsf{ss}}$ since we can use a single variable that marks the first state to test if the path in question is cyclic.

Theorem 3.24. For each Cycle-CTL* formula there is an equivalent formula in H^{1}CTL$^*_{\mathsf{ss}}$ of linear size.

Proof. We only need to discuss how the path quantifier $\mathsf{E}^{\circlearrowleft}$ can be simulated by means of hybrid operators. For this let $\varphi \in$ Cycle-CTL*. We replace each $\mathsf{E}^{\circlearrowleft}\psi$ subformula in φ with $\downarrow x.\mathsf{E}(\mathsf{GF}x \wedge \psi)$. The resulting formula is only linear in the size of φ and it simply simulates the cyclic property at the relevant path quantifications. The same can be done for $\mathsf{A}^{\circlearrowleft}$. $\square$

The addition of cyclic path quantifiers truly increases the expressive power of CTL* [36]. Thus, we obtain that CTL$^* \prec$ Cycle-CTL$^* \preceq$ H^{1}CTL$^*_{\mathsf{ss}}$.
Cycle-CTL* has a PSPACE-complete model checking problem. We will later – in Chapter 6 – discuss the model checking problem for our hybrid logics and prove that HCTL$^*_{\mathsf{ss}}$ also has a PSPACE-complete model checking problem. Paired with the linear translation above this gives an alternative proof and an alternative model checking procedure for Cycle-CTL*.

Counting-CTL* and Graded CTL*. Counting-CTL* [72] is another extension of CTL* that adds a new operator which is able to count the number of successors at a state. The formula $\mathsf{D}^n\varphi$ is satisfied at a state s if and only if there are n successors of s that satisfy φ. A full definition of Counting-CTL* is given in Section 5.3.

Counting the number of successors can also be done with variables. For example the formula $\mathsf{D}^2 p$ for some atomic proposition $p \in \mathit{Prop}$ is equivalent to the formula $\downarrow x.\mathsf{EX}(p \wedge \downarrow y.\, @_x\, \mathsf{EX}(p \wedge \neg y))$ which simply marks the current state and then goes through the successors marking and thereby counting them one-by-one. This idea can easily be generalised to show that Counting-CTL$^* \preceq \mathrm{HCTL}^*_{\mathsf{ss}}$.

We do not show the full translation here, but give it later in Section 5.3. For the moment it suffices to know that the hybrid framework can be used to count successors.

The idea of counting is taken even further by *graded* extensions of CTL or CTL*. These aim not only to count the number of direct successors but in general the number of paths starting at the current state that satisfy a path formula. Syntactically these graded extensions add *graded path quantifiers* of the form $\mathsf{E}^{\geq n}\psi$ with the intuitive meaning "there are at least n different paths satisfying ψ".

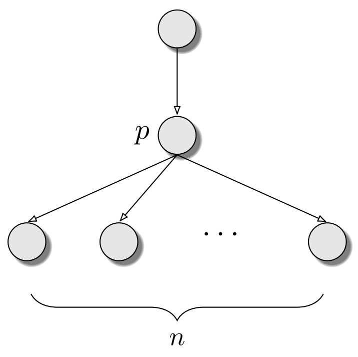
There is however one big difference to simply counting the number of successors as done above with Counting-CTL*. Intuitively, one might think that the formula $\varphi^n := \mathsf{E}^{\geq n}\mathsf{X}p \wedge \neg\mathsf{E}^{\geq(n+1)}\mathsf{X}p$ should then also state that there are n successors satisfying φ. However, the precise meaning of this formula depends in particular on when two (infinite) paths are considered to be different.

The easiest definition that is used for graded logics is to consider two (finite or infinite) paths π_1, π_2 as different iff there is some index $i \in \mathbb{N}$ such that

$$\pi_1^i \neq \pi_2^i,$$

i.e. if they diverge at some point. Formulas of type $\mathsf{E}^{\geq n}\psi$ are thus satisfied at some state iff there are at least n diverging paths satisfying ψ. We call this the diverging paths semantics.

Remark 3.25. In this case the formula φ^n above is also satisfied at the root of the structure



since there are n diverging paths, however, at the root there is only one successor node. Thus with this semantics $\mathsf{graded\ CTL}^*$ is not an extension of $\mathsf{Counting\text{-}CTL}^*$.

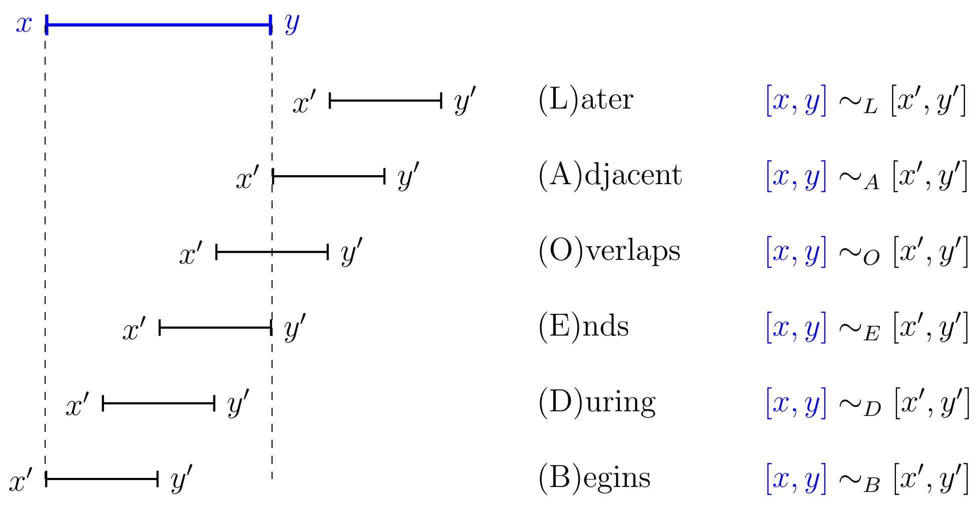
Already for this semantics of different paths it is not obvious how to translate a formula of the form $\mathsf{E}^{\geq 2}\psi$ to our hybrid logics since this would involve finding two paths and a specific moment on both paths such that they diverge. However, it is not clear how to *remember* two paths at the same time with hybrid branching-time logics to compare their states.
But at least on special types of structures this is possible. If we only restrict the logics to be interpreted over trees then this challenge is easily solved since in trees there is exactly one path to a node in the tree. Hence, two paths diverge at some point if and only if there is a node on one path that is not present on the other path. This property can be formulated with hybrid logics. For example, the formula

$$\downarrow root.\mathsf{E}(\psi \wedge \mathsf{F}\downarrow x.\ @_{root}\,\mathsf{E}(\psi \wedge \mathsf{G}\neg x)$$

stating that there is a path satisfying ψ and a moment or state x on this path such that there is another path which never visits x and also satisfies ψ is over trees equivalent to the $\mathsf{graded\ CTL}^*$ formula $\mathsf{E}^{\geq 2}\psi$ with the semantics as described above.
This pattern can also be generalised to simulate the operator $\mathsf{E}^{\geq n}\psi$ as a whole by employing n variables, one for each new path.

Proposition 3.26. On trees and with the diverging paths semantics we have that $\mathsf{graded\ CTL}^* \preceq \mathrm{HCTL}^*_{\mathsf{ss}}$.

Remark 3.27. Another notion of "different paths" that more closely resembles the intended meaning of "there exist n different paths" has also been considered [12, 3] but we will refrain from also comparing this semantics to our hybrid logics. This particular semantics also involves interpretations of the temporal operators over finite paths which are not defined in our hybrid



x
y
x' y' (L)ater $[x,y] \sim_L [x',y']$
x' y' (A)djacent $[x,y] \sim_A [x',y']$
x' y' (O)verlaps $[x,y] \sim_O [x',y']$
x' y' (E)nds $[x,y] \sim_E [x',y']$
x' y' (D)uring $[x,y] \sim_D [x',y']$
x' y' (B)egins $[x,y] \sim_B [x',y']$

Figure 3.4: Allen's interval relations.

branching-time logics. This makes a rigorous comparison hard since already the basic temporal operators have slightly different meanings.
However, it is shown in [3] that graded CTL^* with this semantics on trees is equivalent in terms of expressive power to Monadic Path Logic. In Section 5.3 we will show that also $\mathrm{HCTL}^*_{\mathsf{ss}}$ is equivalent to Monadic Path Logic on trees thus effectively showing that graded $\mathsf{CTL}^* \equiv_{\mathsf{tree}} \mathrm{HCTL}^*_{\mathsf{ss}}$ under this finer semantics.

Interval Temporal Logic. This is a family of logics that was designed with yet another model of time in mind (besides the linear and branching time model). As the name suggests it aims to capture properties that happen at certain *time intervals* rather than at durationless time instants as it is the case for all branching-time logics considered in this thesis.
A prominent logic among these interval-based modal and temporal logics is the Halpern-Shoham logic [45, 73]. This is a multi-modal logic that features one modality for each of Allen's interval relations [2] depicted in Figure 3.4. For example $[x, y] \sim_A [x', y']$ iff $x' = y$ and $x' < y'$ or $[x, y] \sim_B [x', y']$ iff $x' = x$ and $y' < y$. Also it features modalities for the converse relations, denoted by $\overline{X}$ for some interval relation $X \in \{L, A, O, E, D, B\}$ with $[x, y] \sim_{\overline{X}} [x', y']$ if and only $[x', y'] \sim_X [x, y]$.
Usually this logic is interpreted with respect to linear orders, i.e. word structures with a reflexive, transitive transition relation (antisymmetry is already encoded in the word structure) and an interval on this structure. Sometimes

even uncountable structures or structures that are infinite in both directions are considered. Labellings are usually defined by assigning atomic propositions to each interval.
The satisfaction relation is then a straightforward extension of modal logic with $\langle X\rangle\varphi$ meaning that there is an interval that stands in relation X (as depicted in Figure 3.4) with the current interval and that satisfies φ.

Example 3.28. The Halpern-Shoham formula $\langle A\rangle[B]\mathtt{ff} \vee [A][B]\langle B\rangle\mathtt{tt}$ – if satisfied at all states in a linear structure – states that for each interval there is either an adjacent interval which contains no "smaller" intervals or each adjacent interval contains infinitely many "smaller" intervals.
It can be shown ([95]) that this formula defines strictness of intervals in the sense that no "point" intervals $[x, x]$ are allowed.

Of course for our hybrid logics the labelling is defined for single states and not for intervals and thus we are unable to find a translation in either direction. But with hybrid logics we can express these basic interval relations.
On linear orders an interval, instead of a set of connected points, can also be identified with only two points – the start and end point of an interval. This can be simulated by hybrid logics with two variables – one marking the beginning of an interval and one that marks the end.
Allen's interval relations which then consist of only simple relationships between four points in a structure can then simply be simulated with jumps and temporal operators. For example $[x, y] \sim_A [x', y']$ iff $x' = y$ and $x' < y'$. Thus, we can translate a formula $\langle A\rangle\varphi$ as follows:

$$\langle A\rangle\varphi \qquad \rightsquigarrow \qquad @_y \downarrow x.\mathsf{EX}(\neg x \wedge \downarrow y.\widehat{\varphi}).$$

The first jump and binder simply move the x-variable used to mark the beginning of an interval to the state marked y and then move on to a later state which is checked by $\neg x$ and then place y there.
Similar translations are possible for the other interval relations:

$$\begin{array}{lll}
\langle L\rangle\varphi & \rightsquigarrow & @_y\,\mathsf{EX}(\neg y \wedge \downarrow x.\mathsf{EX}(\neg x \wedge \mathsf{EX}\downarrow y.\widehat{\varphi})), \\
\langle O\rangle\varphi & \rightsquigarrow & @_x\,\mathsf{EX}((\mathsf{EX}y) \wedge \downarrow x.\mathsf{EX}(\neg(\mathsf{EX}y) \wedge \downarrow y.\widehat{\varphi})), \\
\langle E\rangle\varphi & \rightsquigarrow & @_x\,\mathsf{EX}((\mathsf{EX}y) \wedge \downarrow x.\widehat{\varphi}), \\
\langle D\rangle\varphi & \rightsquigarrow & @_x\,\mathsf{EX}((\mathsf{EX}y) \wedge \downarrow x.\mathsf{EX}((\mathsf{EX}y) \wedge \downarrow y.\widehat{\varphi})), \\
\langle B\rangle\varphi & \rightsquigarrow & @_x\,\mathsf{EX}((\mathsf{EX}y) \wedge \downarrow y.\widehat{\varphi}).
\end{array}$$

For the converse relations we have the problem that some of the intervals actually lie *before* the current interval. Since we only have future operators and no past operators we can only simulate them if the structure is rooted,

i.e. it is only infinite in the future direction. Then we can place a nominal at the root and use similar constructions as above with the root helping us to "jump" back in time.
Thus, we can express Halpern-Shoham formulas that make no use of atomic propositions, i.e. that have only $\mathtt{tt}, \mathtt{ff}$ as atomic formulas and no converse relations. And if the structure is only infinite in the future direction we can even express Halpern-Shoham formulas with converse relations .

Proposition 3.29. For each HALPERN-SHOHAM formula φ with no atomic propositions and no converse relations there is a formula $\widehat{\varphi} \in \text{HCTL}$ that is linear in the size of φ such that $\mathcal{K}, [s,t] \models \varphi$ if and only if $\mathcal{K}, s, \sigma \models \widehat{\varphi}$ for all states $s \in S$ and all variable assignments $\sigma : \mathit{Var} \to S$ such that $\sigma(x) = s$ and $\sigma(y) = t$.
On linear orders that are only infinite in the future direction there is a similar translation available for all HALPERN-SHOHAM formulas (including converse interval relations) with no atomic propositions.

As shown above in Example 3.28 interval-based logics even without the use of atomic propositions and thus also hybrid logics can express quite non-trivial properties.

Quantified CTL*. This logic – also called QCTL^*– extends CTL* with additional full quantification over atomic propositions [66]. Formally, it adds the clause $\exists p.\varphi$ to the state formulas which is satisfied in a state s of a Kripke structure $\mathcal{K}$ iff there exists a Kripke structure $\mathcal{K}'$ which only differs in the labeling of p from $\mathcal{K}$ such that $\mathcal{K}'$ satisfies p.
Full quantification over propositions is a powerful tool as it is essentially a second-order quantifier which can mark sets of states. Thus it is not too surprising that the expressive power of QCTL^* coincides – at least on finite structures and tree structures – with MSO that is restricted to only evaluate the reachable part of a structure [66]. In fact, already quantified CTL is as expressive as MSO.
Quantified CTL* is also able to express many of the previously shown examples like cyclic behaviour or counting of successors. In fact, it can quite simply simulate the binder as a state formula. For example, the formula $\forall x.(x \to \varphi)$ means that for all labelings with a proposition x – in particular the labeling where x is only bound to the current state – if x holds now then φ must be satisfied. Hence, this formula has the same meaning as $\downarrow x.\varphi$.
However, it is not obvious how the jump operator should be mimicked since QCTL^* has no similar operator for this. Also, jumps can potentially lead to states that are not directly reachable anymore with only future operators.

For a variable might be placed at the root of a tree at the start but with only future operators in QCTL^* it is not possible to go back to the root while this is easily possible with jumps.
This does not mean that it is not possible, but it is simply not obvious how to do it. In fact, on trees we will later show that also $\mathrm{HCTL}^*_{\mathsf{ss}}$ coincides with Monadic Path Logic which is MSO where the second-order quantification is restricted to paths in a tree only. Thus, we get that, at least on trees, $\mathrm{HCTL}^*_{\mathsf{ss}} \preceq_{\mathsf{tree}} \mathsf{QCTL}^*$. For finite structures or general Kripke structures the precise connection is not yet clear.

Chapter 4

The Expressive Power of the Fully Hybrid μ-calculus

Hybridisation of branching-time logics adds considerable expressive power to them. For example, we know that all temporal logics from CTL up to the μ-calculus can only express bisimulation-invariant properties. It is easily seen that hybridisation breaks these limitations. For example simply by naming one state one can distinguish it from an otherwise bisimilar state. However, we do not really know what types of properties *can* be expressed with these hybrid logics. Furthermore, it is not clear where the limits of these newly added hybrid operators are or which types of properties *cannot* be expressed with these logics.

In this chapter we study these issues. Specifically, we will take a look at the fully hybrid μ-calculus and try to better understand its expressive power and the limits of it. Although we only deal with the fully hybrid μ-calculus at first, the model theoretic results that will be proven in Chapter 5 will transfer many of our findings in this chapter to the other hybrid branching-time logics as well.

The remaining chapter is organised as follows. We begin with a comparison to another well-known extension of the modal μ-calculus – the polyadic μ-calculus – to get a better impression of what can be expressed with the hybrid μ-calculus. We then introduce a suitable notion of bisimulation that precisely captures the expressive power of the hybrid μ-calculus and use it to exhibit some of the limitations of the hybrid μ-calculus. We finish the chapter with an observation about the bounded fragments of H_μ that applies some of the previous results.

4.1 The Polyadic μ-calculus and H_μ

The polyadic μ-calculus, or higher-dimensional μ-calculus [4, 75, 64], is another extension of the modal μ-calculus. But rather than specifying properties that are satisfied at a single state of the structure, the polyadic μ-calculus is designed to express relational properties between states of a structure. Consequently, formulas of the polyadic μ-calculus are interpreted in tuples of states.

Formally the polyadic μ-calculus of arity k – or short L^k_μ – is given by the grammar

$$\varphi := p(i) \mid X \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box_i\varphi \mid \mu X.\varphi(X) \mid \{i \leftarrow j\}\varphi.$$

for $p \in \mathit{Prop}$, $X \in \mathit{Var}_2$, $i, j \in \{1, \ldots, k\}$. The usual restrictions on fixed points apply here in the same way as before.

The semantics are given inductively with respect to a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ and an assignment $\rho : \mathit{Var}_2 \rightarrow 2^{S^k}$ as follows:

$$\begin{aligned}
[\![p(i)]\!]^{\mathcal{K}}_\rho &= \{(s_1, \ldots, s_k) \in S^k \mid s_i \in L(p)\},\\
[\![X]\!]^{\mathcal{K}}_\rho &= \rho(X),\\
[\![\neg\varphi]\!]^{\mathcal{K}}_\rho &= \{(s_1, \ldots, s_k) \in S^k \mid (s_1, \ldots, s_k) \notin [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}}_\rho &= [\![\varphi_1]\!]^{\mathcal{K}}_\rho \cup [\![\varphi_2]\!]^{\mathcal{K}}_\rho,\\
[\![\Box_i\varphi]\!]^{\mathcal{K}}_\rho &= \{(s_1, \ldots, s_k) \in S^k \mid \forall t \in S\text{: if } s_i \rightarrow t,\\
&\qquad \text{then } (s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\},\\
[\![\mu X.\varphi(X)]\!]^{\mathcal{K}}_\rho &= \bigcap\{T \subseteq S^k \mid [\![\varphi]\!]^{\mathcal{K}}_{\rho[X\rightarrow T]} \subseteq T\},\\
[\![\{i \leftarrow j\}\varphi]\!]^{\mathcal{K}}_\rho &= \{(s_1, \ldots, s_k) \in S^k \mid (s_1, \ldots, s_{i-1}, s_j, s_{i+1}, \ldots, s_k) \in [\![\varphi]\!]^{\mathcal{K}}_\rho\}.
\end{aligned}$$

The polyadic μ-calculus L^ω_μ is given by the union of all L^k_μ.

Remark 4.1. In [75] or [64], replacements are defined by simultaneously substituting multiple variables instead of only a single one as defined here. For example the replacement $\{1 \leftarrow 2, 2 \leftarrow 1\}$ swaps the variables 1 and 2.

Our definition does not change anything about the expressive power, however, to express some properties like swapping variables as above we may need an additional dimension to temporarily store the variables.

The *arity* of a formula is the largest index i that occurs in the operators $\Box_i, p(i), \{i \leftarrow j\}$ in this particular formula. Note that a formula of arity k defines a k-ary relation over the set of states. However it can also be seen as a relation of higher arity where the $(k + 1), (k + 2)$, etc. components are unrestricted.

We write $\mathcal{K}, (s_1, \ldots, s_k), \rho \models \varphi$ if $(s_1, \ldots, s_k) \in [\![\varphi]\!]^{\mathcal{K}}_{\rho}$ and may drop ρ for closed formulas.
The usual modal abbreviations like $\Diamond_i \psi := \neg\Box_i\neg\psi$ etc. are also used here.

Example 4.2. The polyadic μ-calculus is a very expressive logic. For example it is able to express *bisimilarity* of two states:

$$\nu Y. \left((\bigwedge_{p \in Prop} p(1) \leftrightarrow p(2)) \wedge (\Box_1 \Diamond_2 Y) \wedge (\Box_2 \Diamond_1 Y) \right).$$

If we compare this formula with the definition of bisimularity in Definition 2.5, it is easy to see that this formula simply simulates all clauses in the definition of bisimulation: two states agree on all propositions and for every successor in one structure there is a successor in the other structure such that both successors are bisimilar which here can be expressed by being a part of the fixed point again. Thus the formula is satisfied on a pair of states (s_1, s_2) if and only if $s_1 \sim s_2$.

H_μ and L^ω_μ seem closely related. Formulas of L^k_μ define k-ary relations, while formulas of H^{k-1}_μ define a set of tuples consisting of a state and a variable interpretation that references $k-1$ states. This of course can also be regarded as a k-ary relation. Since both logics build upon the same foundation, namely L_μ, they share most operators.
In the remainder of this section we will prove that H_μ subsumes the polyadic μ-calculus and we will also show what is lacking in the polyadic μ-calculus to achieve the same expressiveness as H_μ, hence, making the inclusion strict.

Theorem 4.3. For every formula $\varphi \in \mathrm{L}^k_\mu$ there is a formula $\psi \in \mathrm{H}^k_\mu$ such that for all Kripke structures $\mathcal{K} = \langle S, \rightarrow, L \rangle$, $s \in S$, $(s_1, \ldots, s_k) \in S^k$ and all variable assignments $\sigma : Var \rightarrow S$ with $\sigma(x_i) = s_i$ it holds that

$$\mathcal{K}, (s_1, \ldots, s_k) \models \varphi \quad \Leftrightarrow \quad \mathcal{K}, s, \sigma \models \psi.$$

Proof. We will give a translation from L^k_μ to H^k_μ. The idea is simple: we simulate the k-tuple using k variables. Thus each variable corresponds to an index in φ. The "current" state that is additionally present in H^k_μ together with the hybrid operators will only be used to simulate temporal steps and replacements.
Formally, the translation is given inductively as follows:

$$\tau(p(i)) = @_{x_i} p,$$
$$\tau(X) = X,$$

$$\begin{aligned}
\tau(\neg\varphi) &= \neg\tau(\varphi),\\
\tau(\varphi_1 \vee \varphi_2) &= \tau(\varphi_1) \vee \tau(\varphi_2),\\
\tau(\Box_i\varphi) &= @_{x_i} \Box \downarrow x_i.\tau(\varphi),\\
\tau(\mu X.\varphi) &= \mu X.\tau(\varphi),\\
\tau(\{i \leftarrow j\}\varphi) &= @_{x_j} \downarrow x_i.\tau(\varphi).
\end{aligned}$$

To prove that this translation works, we first need to extend the statement to account for free second-order variables. Thus, we prove the following statement.
For every $\varphi \in \mathrm{L}_\mu^k$, every Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ and every $(s_1, \dots, s_k) \in S^k$, $\sigma : \mathit{Var} \rightarrow S$, $\rho : \mathit{Var}_2 \rightarrow 2^{S^k}$, $\rho' : \mathit{Var}_2 \rightarrow S \times (\mathit{Var} \rightarrow S)$ such that $\sigma(x_i) = s_i$ for every $i \in \{1, \dots, k\}$ and $\rho'(X) = \{(s, \sigma) \mid s \in S, (\sigma(x_1), \dots, \sigma(x_k)) \in \rho(X)\}$ we have that

$$\mathcal{K}, (s_1, \dots, s_k), \rho \models \varphi \quad \Leftrightarrow \quad \mathcal{K}, (s, \sigma), \rho' \models \tau(\varphi).$$

This can be proven by a straightforward induction on φ. The details are omitted, we give only a brief description on the ideas needed for the actual proof.
Note that the actual state on which the hybrid formula is evaluated is irrelevant by construction of ρ' and the fact that all translated formulas – except for boolean combinations or least fixed point constructs – begin with a jump. Thus, either all states or no state satisfies such a formula. The cases for box and replacement simply use the "current" state in the evaluation to go to one of the referenced states and then carry out the actual operation there before rebinding the variable to the new state. The case for smallest fixed points can be proven by another induction on the fixed point approximations. $\Box$

A closer inspection of the translation from L_μ^ω to H_μ reveals that the translated formula even falls into $\mathrm{H}_\mu^k(\downarrow, @)$ – the fragment which does not allow variable tests. This is not surprising because the hard requirement that the variable test x is only satisfied at exactly the state $\sigma(x)$ is what breaks bisimulation-invariance and the polyadic μ-calculus is still known to respect bisimulation [75].
Variable tests can be seen as a restricted kind of equality: The current state of the evaluation needs to be *equal* to the stored state. We will now show that a restricted form of "equality" is also the only thing that is missing in L_μ^ω to be as expressive as H_μ. Let $\mathrm{L}_\mu^k(\doteq)$ be the logic that is obtained by also adding the statement $i \doteq j$ for $i, j \in \{1, \dots, k\}$ to the grammar of L_μ^k and $\mathrm{L}_\mu^\omega(\doteq)$ be the union of all $\mathrm{L}_\mu^k(\doteq)$. Formally, this statement is interpreted as

follows:

$$[\![i \doteq j]\!]^{\mathcal{K}}_{\rho} = \{(s_1, \ldots, s_k) \in S^k \mid s_i = s_j\}.$$

By extending the translation from Theorem 4.3 above by $\tau(i \doteq j) = @_{x_i} x_j$ we can easily see that $\mathrm{L}^{\omega}_{\mu}(\doteq)$ is still subsumed by H_{μ}. But for this extended logics we can now also prove the reverse.

Theorem 4.4. For every formula $\varphi \in \mathrm{H}^k_\mu$ there is a formula $\psi \in \mathrm{L}^{k+1}_\mu(=)$ such that for any Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ and any assignment σ with $\sigma(x_i) = s_i$ for all $i = 1, \ldots, k$ it holds that

$$\mathcal{K}, s, \sigma \models \varphi \quad \Leftrightarrow \quad \mathcal{K}, (s_1, \ldots, s_k, s) \models \psi.$$

Proof. Again, we give a translation, this time from H^k_μ to $\mathrm{L}^{k+1}_\mu(\doteq)$. The idea is to use the first k components of the $\mathrm{L}^{k+1}_\mu(\doteq)$ formula simply to store the variables and the last component is used to simulate the temporal operators. Replacements are used to simulate the hybrid operators @ and $\downarrow$ and the new equality statement can be used to simulate variable tests between the last component and the components that store the states.
Formally, the translation on a H^k_μ formula is given inductively as follows:

$$\begin{aligned}
\tau(p) &= p(k+1)\\
\tau(x_i) &= ((k+1) \doteq i)\\
\tau(X) &= X\\
\tau(\neg\varphi) &= \neg\tau(\varphi)\\
\tau(\varphi_1 \vee \varphi_2) &= \tau(\varphi_1) \vee \tau(\varphi_2)\\
\tau(\Box\varphi) &= \Box_{k+1}\tau(\varphi)\\
\tau(@_{x_i}\varphi) &= \{(k+1) \leftarrow i\}\tau(\varphi)\\
\tau(\downarrow x_i.\varphi) &= \{i \leftarrow (k+1)\}\tau(\varphi)\\
\tau(\mu X.\varphi) &= \mu X.\tau(\varphi).
\end{aligned}$$

Correctness of the translation can be proven by a straightforward induction on φ similar to the proof of Theorem 4.3. The details are also omitted. $\Box$

A closer inspection of the translation reveals that the introduced equality is only used in the case of variable tests. Thus we also get the following result.

Corollary 4.5. $\mathrm{H}^k_\mu(\downarrow, @) \preceq \mathrm{L}^{k+1}_\mu$

Combining the previous results we obtain the following result about the logics without restrictions on the number of variables.

Corollary 4.6. The polyadic μ-calculus L_μ^ω and $H_\mu(\downarrow, @)$ have the same expressive power and there are linear translations between both logics.

Using the translation in Theorem 4.3 it is now easy to see that H_μ is for example able to express bisimilarity of two states.

Example 4.7. The H_μ formula

$$\begin{aligned}\varphi_\sim \;\; := \;\; \nu X.\Big(\big(\bigwedge_{p \in Prop} @_x\, p \leftrightarrow @_y\, p\big) \wedge (@_x \Box \downarrow x.\, @_y \Diamond \downarrow y.X) \\ \wedge (@_y \Box \downarrow y.\, @_x \Diamond \downarrow x.X)\Big)\end{aligned}$$

"expresses bisimulation" – similar to the L_μ^2 formula in Example 4.2. To be more precise it holds that $\mathcal{K}, s, \sigma \models \varphi_\sim$ on a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ if and only $\sigma(x) \sim \sigma(y)$. This can be seen by either using the translations from L_μ^2 to H_μ^2 in Theorem 4.3 or via a direct proof that utilises the model checking games for H_μ presented in Section 3.5.3. Observe that strategies in the model checking game directly correspond to strategies in the bisimulation game for the respective players.

4.2 Hybrid Bisimulation

Bisimulation is the key notion for many decidability results and decision procedures with regard to temporal logics. Most temporal logics from CTL up to the μ-calculus are invariant under bisimulation, i.e. they cannot distinguish two bisimilar states.
On the other hand, the notion of bisimilarity also characterises the expressive power of temporal logics in some sense. It gives an upper bound on which structures can be distinguished with a logic and which structures can be regarded as equivalent with respect to the logic. Hence, for many reasons such a notion is very useful.
However, as already stated previously, it is easy to see that this is not the case anymore for hybrid logics. For example the hybrid modal logic formula $\downarrow x.\Diamond x$ – which checks if there is a loop – can distinguish a state from its tree unwinding. Thus the usual bisimulation notion is not fine enough to capture the expressive power of hybrid logics.
For this reason we recall a finer notion of bisimulation – called k-bisimulation – which is tailored to capture the expressive power of hybrid logics and was already introduced in [7] to show that hybrid modal logic with at most k variables is invariant under this refined form of bisimulation. We will extend this result and show that the same is true for H_μ^k.

Let $\sim^k \subseteq (S_0 \times S_0^k) \times (S_1 \times S_1^k)$ be a $2k+2$-ary relation over a set of states S_0 and S_1. To enhance readability we will denote $\sim^k$ as a binary relation relating tuples $(s, s_1, \ldots, s_k) \in S_0 \times S_0^k$ and $(t, t_1, \ldots, t_k) \in S_1 \times S_1^k$ and will also denote tuples $(s_1, \ldots, s_k)$ by $\overline{s}$ if k is clear from the context.

Definition 4.8. Let $k \in \mathbb{N}$ and let $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0 \rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1 \rangle$ be two Kripke structures.
A *k-bisimulation* is a non-empty relation $\sim^k \subseteq (S_0 \times S_0^k) \times (S_1 \times S_1^k)$ such that for all $(s, \overline{s}) \sim^k (t, \overline{t})$ we have:

(prop) for all $p \in \mathit{Prop}$ it holds that $s \in L_0(p)$ if and only if $t \in L_1(p)$ and for all $n \in \mathit{Nom}$ it holds that $s = L_0(n)$ if and only if $t = L_1(n)$,

(var) for all $i \leq k$ it holds that $s = s_i$ if and only if $t = t_i$,

(zig) for every $s' \in S_0$ such that $s \rightarrow_0 s'$ there is a $t' \in S_1$ with $t \rightarrow_1 t'$ such that $(s', \overline{s}) \sim^k (t', \overline{t})$,

(zag) for every $t' \in S_1$ such that $t \rightarrow_1 t'$ there is a $s' \in S_0$ with $s \rightarrow_0 s'$ such that $(s', \overline{s}) \sim^k (t', \overline{t})$,

(nom) for every $n \in \mathit{Nom}$ it holds that $(L_0(n), \overline{s}) \sim^k (L_1(n), \overline{t})$,

(@) for every $i \leq k$ it holds that $(s_i, \overline{s}) \sim^k (t_i, \overline{t})$ and

($\downarrow$) for every $i \leq k$ it holds that $(s, s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_k) \sim^k (t, t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_k)$.

We say that $s \sim^k t$ iff $(s, s, \ldots, s) \sim^k (t, t, \ldots, t)$ and we say that $\mathcal{K}_0 \sim^k \mathcal{K}_1$ iff there are states $s, s_1, \ldots, s_k \in S_0$ and states $t, t_1, \ldots, t_k \in S_1$ such that $(s, \overline{s}) \sim^k (t, \overline{t})$.

As for the usual bisimulation notion, k-bisimulation can also be thought of as a game played by two players in which one player tries to distinguish both structures and the other one tries to show that this is not possible.

Definition 4.9. Let $k \in \mathbb{N}$ and let $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0 \rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1 \rangle$ be two Kripke structures over the same set of atomic propositions *Prop* and nominals *Nom*.
The *k-bisimulation game* $\mathcal{G}^k(\mathcal{K}_0, s, \overline{s}, \mathcal{K}_1, t, \overline{t})$ is played between **Spoiler** and **Duplicator** on the configuration space $(S_0 \times S_0^k) \times (S_1 \times S_1^k)$.
The game starts with two *active* pebbles P^0 and P^1 placed on s and t respectively and $2k$ *passive* pebbles $P_1^0, \ldots, P_k^0$ and $P_1^1, \ldots, P_k^1$ placed on the states $s_1, \ldots, s_k$ in $\overline{s}$ resp. $t_1, \ldots, t_k$ in $\overline{t}$.

The state on which a pebble Q lies is referred to as $\mathsf{st}(Q)$.
Both players take turns beginning with **Spoiler**. He begins by choosing one structure $\mathcal{K}_i$ for some $i \in \{0,1\}$ and then chooses one of the following moves:

- choose a transition $\mathsf{st}(P^i) \rightarrow_i s'$ and move P^i from $\mathsf{st}(P^i)$ to s'.
- choose a passive pebble P^i_j, $j \in \{1,\dots,k\}$ and move it to $\mathsf{st}(P^i)$.
- choose a passive pebble P^i_j, $j \in \{1,\dots,k\}$ and move P^i to $\mathsf{st}(P^i_j)$.
- choose a nominal $n \in \mathit{Nom}$ that occurs in $\mathcal{K}_i$ and move P^i to $L_i(n)$.

After that **Duplicator** is forced to make the same kind of move but on the opposite structure $\mathcal{K}_{1-i}$.
Spoiler wins, if after **Duplicator**'s move the game is in a position such that:

- there is a $p \in \mathit{Prop}$ such that $\mathsf{st}(P^0) \in L_0(p)$ but $\mathsf{st}(P^1) \notin L_1(p)$ or vice versa,
- there is an $n \in \mathit{Nom}$ such that $\mathsf{st}(P^0) = L_0(n)$ but $\mathsf{st}(P^1) \neq L_1(n)$ or vice versa, or
- there is $j \in \{1,\dots,k\}$ such that $\mathsf{st}(P^0) = \mathsf{st}(P^0_j)$ but $\mathsf{st}(P^1) \neq \mathsf{st}(P^1_j)$ or vice versa.

On the other hand, Duplicator only wins if she can successfully match all the atomic propositions, nominals and pebbles after each of **Spoiler**'s moves and the game continues forever.

It is not too difficult to see that Definitions 4.8 and 4.9 lead to the same notion of k-Bisimulation:

Proposition 4.10. Let $k \in \mathbb{N}$ and let $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0\rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1\rangle$ be two Kripke structures. There is a k-bisimulation between $\mathcal{K}_0$ and $\mathcal{K}_1$ with $(s,\overline{s}) \sim^k (t,\overline{t})$ if and only if Duplicator wins $\mathcal{G}^k(\mathcal{K}_0, s, \overline{s}, \mathcal{K}_1, t, \overline{t})$.

Proof. (Sketch) To see this simply note that the moves in the k-Bisimulation game correspond to the clauses (zig) up to ($\downarrow$) in Definition 4.8 and the winning conditions after each round in the game correspond to the remaining clauses (prop) and (var). With this observation it is easy to construct a winning strategy for Duplicator in the k-bisimulation game on k-bisimilar structures given that there is a k-bisimulation between the structures and vice versa. $\square$

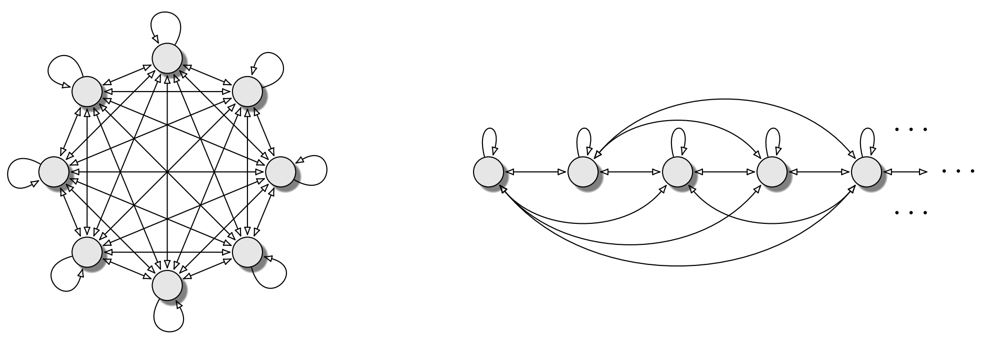


Figure 4.1: C_8 and C_∞.

k-Bisimulations can easily be restricted to fit the underlying logic. For example, the hybrid μ-calculus introduced by Sattler and Vardi in [80] only features nominals and jumps but no $\downarrow$-operation to dynamically name states. To get the corresponding bisimulation notion we simply dismiss the ($\downarrow$) and (*var*) clause above or the respective moves in the k-bisimulation game, essentially going back to a binary relation with $k = 0$. Restricting the clauses even further and also dismissing (*nom*) and (@) then simply gives us the usual bisimulation notion that captures for example the expressive power of the modal μ-calculus.
To better understand this relation we give an example of k-bisimulation. Let $C_k = \langle S_k, \to, L\rangle$ be the complete clique over k states and $C_\infty = \langle S_\infty, \to, L\rangle$ be the complete clique over $\mathbb{N}$. Figure 4.1 depicts C_8 (left) and C_∞ (right) as Kripke structures over $\mathit{Prop} = \mathit{Nom} = \emptyset$.

Lemma 4.11. *$C_{k+1} \not\sim^k C_j$ for all $j \in \{1, \dots, k\}$.*

Proof. We prove this by giving a winning strategy for Spoiler in the k-bisimulation game on C_{k+1} and C_j for some $j \leq k$.
We describe Spoiler's winning strategy starting at a configuration $(s, s, \dots, s, t, t, \dots, t)$. It is irrelevant which structure Spoiler chooses. In each turn he simply takes one transition and moves the active pebble to a nonmarked location. In the following turn he moves one of the previously not used passive pebbles to his new location. Duplicator will always do the same. Spoiler does this for $2(j-1)$ turns. After that round all states in C_j are marked with one passive pebble. In this round he simply chooses to move on C_{k+1} and moves to an unmarked location. Since $j < k+1$ there is at least one unmarked state in C_{k+1} left. Duplicator then has to move on C_j and since all states are marked Duplicator has to move to a marked state and loses the game. □

Lemma 4.12. *$C_{k+1} \sim^k C_j$ for all $j > k+1$. Especially $C_{k+1} \sim^k C_\infty$.*

Proof. Let $j, k \in \mathbb{N}$ such that $j > k+1$. We will show the lemma by describing a winning strategy for Duplicator in the k-bisimulation game on C_{k+1} and C_j.
We call a configuration $(s, s_1, \dots, s_k, t, t_1, \dots, t_k)$ *consistent* if for all i we have $s = s_i$ if and only if $t = t_i$. Notice that since no atomic propositions or nominals are present in C_{k+1} and C_j, Spoiler only wins in a configuration that is not consistent.
This induces a simple winning strategy for Duplicator. She simply needs to preserve consistency. It should be clear that with such a strategy, any game starting in a consistent configuration like $(s, s, \dots, s, t, t, \dots, t)$ is then won by Duplicator and hence, the lemma is proven.
So, let $(s, s_1, \dots, s_k, t, t_1, \dots, t_k)$ be a consistent configuration. If Spoiler moves to one of the passive pebbles, or moves one of the passive pebbles to the active one, Duplicator simply does the same thing on the second structure. It is not hard to see that the configuration after Duplicator's move is again consistent. So, assume that Spoiler moves the active pebble on, for example, s to some successor state s'. There are two cases. Either s' is already marked with one of the passive pebbles. In this case Duplicator mimics his move and and moves her active pebble from t to the state marked with the corresponding passive pebble. Again, this configuration is consistent. Lastly, suppose s' is not marked by any passive pebble. In both structures there is at least one state that is not marked by any passive pebble because both C_{k+1} and C_j have more than k states. Duplicator chooses one such state t' on her structure and since both structures are cliques this state is also reachable from t and she moves her active pebble to t'. Since both states s' and t' were not marked, consistency is preserved and thus Duplicator wins the game starting at a consistent configuration.
The same strategy holds for the case of C_∞ since this strategy only requires at least one unmarked state in each round. □

Thus, with k-bisimulations we can intuitively distinguish up to k copies of otherwise almost identical states. It is easy to see that for usual bisimulation it already holds that $C_1 \sim C_k$ for every $k \in \mathbb{N}$ or $k = \infty$. Hence, k-bisimulations are much more refined than ordinary bisimulations.

4.3 H_μ and Hybrid Bisimulation

We will now show that k-bisimulation is a good notion that captures the expressive power of H_μ using k variables, i.e. H_μ^k in the following sense.

Theorem 4.13. Let $\varphi \in \mathrm{H}_\mu^k$ be closed and $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0\rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1\rangle$ be two Kripke structures. If $s \in S_0$ and $t \in S_1$ such that $s \sim^k t$ then it holds that $\mathcal{K}_0, s \models \varphi$ if and only if $\mathcal{K}_1, t \models \varphi$.

Proof. Without loss of generality we assume that φ uses variables $x_1, \ldots, x_k$. We will prove a stronger statement that also accounts for free first- and second-order variables:
Let $\rho : \mathit{Var}_2 \rightarrow 2^{S_0 \times (\mathit{Var} \rightarrow S_0)}, \rho' : \mathit{Var}_2 \rightarrow 2^{S_1 \times (\mathit{Var} \rightarrow S_1)}$ be such that they respect k-bisimilarity, i.e. for all $(s, \sigma) \in 2^{S_0 \times (\mathit{Var} \rightarrow S_0)}$ and $(t, \sigma') \in 2^{S_1 \times (\mathit{Var} \rightarrow S_1)}$ and $X \in \mathit{Var}_2$ it holds that if $(s, \sigma(x_1), \ldots, \sigma(x_k)) \sim^k (t, \sigma'(x_1), \ldots, \sigma'(x_k))$ then $(s, \sigma(x_1), \ldots, \sigma(x_k)) \in \rho(X)$ if and only if $(t, \sigma'(x_1), \ldots, \sigma'(x_k)) \in \rho'(X)$.
We prove that if $(s, \sigma) \in 2^{S_0 \times (\mathit{Var} \rightarrow S_0)}$ and $(t, \sigma') \in 2^{S_1 \times (\mathit{Var} \rightarrow S_1)}$ such that $(s, \sigma(x_1), \ldots, \sigma(x_k)) \sim^k (t, \sigma'(x_1), \ldots, \sigma'(x_k))$ then $\mathcal{K}_0, s, \sigma, \rho \models \varphi$ if and only if $\mathcal{K}_1, t, \sigma', \rho' \models \varphi$.
So, assume that $s, t, \sigma, \sigma', \rho, \rho'$ are as stated above. We argue with Definition 4.8 in this proof.
For the base cases, observe that the case for $\varphi = p$ follows directly from the (*prop*) clause of Definition 4.8 stating that k-bisimilar states agree on all the atomic propositions. The case for $\varphi = x$ follows directly from the (Var) clause. And the case $\varphi = X$ for some $X \in \mathit{Var}_2$ follows with the assumption on ρ and ρ' above.
Boolean combinations follow with simple semantical considerations as usual. The case for $\varphi = \Diamond\psi$ follows directly from the (*zig*) and (*zag*) clauses. The hybrid cases $\varphi = \downarrow x.\psi$ resp. $\varphi = @_x\,\psi$ follow immediately with the clauses $(\downarrow)$, $(@)$.
Lastly, suppose that $\varphi = \mu X.\psi(X)$. For this case, we prove via transfinite induction on α, that for all $(s, \sigma) \sim^k (t, \sigma')$ and all ρ, ρ' that satisfy the condition on second-order variable assignments stated above it holds that $\mathcal{K}_0, s, \sigma, \rho \models \mu X^\alpha.\psi(X)$ if and only if $\mathcal{K}_1, t, \sigma', \rho' \models \mu X^\alpha.\psi(X)$.
For the base case $\alpha = 0$ observe that

$$\mathcal{K}_0, s, \sigma, \rho \models \mu X^0.\psi(X) \quad \Leftrightarrow \quad \mathcal{K}_1, t, \sigma', \rho' \models \mu X^0.\psi(X)$$

because $\mu X^0.\psi(X) \equiv \mathtt{ff}$.
First, suppose α is a successor ordinal, hence $\alpha = \beta + 1$ for some ordinal β. We have $\mathcal{K}_0, s, \sigma, \rho \models \mu X^\alpha.\psi(X)$ if and only if $\mathcal{K}_0, s, \sigma, \rho[X \mapsto \mu X^\beta.\psi(X)] \models \psi(X)$. Now observe, that by the induction hypothesis for $\mu X^\beta.\psi(X)$ we get that $\rho[X \mapsto \mu X^\beta.\psi(X)]$ and $\rho'[X \mapsto \mu X^\beta.\psi(X)]$ still respect k-bisimilarity and thus we can use the induction hypothesis for $\psi(X)$ and get that $\mathcal{K}_1, t, \sigma', \rho'[X \mapsto \mu X^\beta.\psi(X)] \models \psi(X)$ and hence $\mathcal{K}_1, t, \sigma', \rho' \models \mu X^\alpha.\psi(X)$.

Lastly, suppose that α is a limit ordinal. Thus, $\mu X^{\alpha}.\psi(X) = \bigcup_{\beta<\alpha} \mu X^{\beta}.\psi(X)$. The claim follows since every $\mu X^{\beta}.\psi(X)$ respects the claim by the induction hypothesis. $\square$

In other words, if two states are k-bisimilar then there is no formula with k variables that is able to distinguish these states. A first application of this so-called k-bisimulation-invariance is the following.

Corollary 4.14. Again, let $C_k = \langle S_k, \to, L\rangle$ be the complete clique over k states and $C_\infty = \langle S_\infty, \to, L\rangle$ be the complete clique over $\mathbb{N}$. There is no formula $\varphi \in \mathrm{H}_\mu$ such that $C_k \models \varphi$ for all $k \in \mathbb{N}$ and $C_\infty \not\models \varphi$.

Proof. For the sake of contradiction, suppose there was such a formula φ. Then $\varphi \in \mathrm{H}^k_\mu$ for some $k \in \mathbb{N}$. By Lemma 4.12 we get that $C_{k+1} \sim^k C_\infty$ and by Theorem 4.13 we get that $C_{k+1} \models \varphi$ if and only if $C_\infty \models \varphi$. This is a contradiction to the assumption on φ. $\square$

We will come back to these results in Chapter 5 in which we provide a connection between H_μ and the other hybrid branching-time logics. This connection will also show which of the other hybrid branching-time logics are invariant under k-bisimulation. For this reason we will refrain from looking further into this right now.
There is another interesting connection between H_μ and k-bisimulations. As we have already seen, H^2_μ is able to express bisimilarity or 0-bisimilarity, i.e. there is a formula with only two variables that is true in a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ if and only if both variables are assigned to bisimilar states. Using more variables it is also easy to see that H_μ can express k-bisimilarity for any $k \in \mathbb{N}$.
For technical convenience we assume that $\mathit{Nom} = \emptyset$ because the usual reduction from bisimilarity of two states in two Kripke structures to bisimulation of two states in one Kripke structure involves collapsing both structures into one. In the presence of nominals that share the same names this would mean we would have to invest some extra work and possibly rename some of them.

Example 4.15. The formula

$$\begin{aligned}\varphi^k_\sim \;:=\; \nu X.\Big(\bigwedge_{p\in \mathit{Prop}} (@_x\, p \leftrightarrow @_y\, p) \wedge \bigwedge_{i=1}^{k} (@_x\, x_i \leftrightarrow @_y\, y_i) \\ \wedge\; (@_x \Box {\downarrow} x.\, @_y \Diamond {\downarrow} y. X) \\ \wedge\; (@_y \Box {\downarrow} y.\, @_x \Diamond {\downarrow} x. X) \\ \wedge\; \bigwedge_{i=1}^{k} (@_{x_i} {\downarrow} x.\, @_{y_i} {\downarrow} y. X)\end{aligned}$$

$$\wedge \bigwedge_{i=1}^{k} (@_x \downarrow x_i.\, @_y \downarrow y_i.X)\Big)$$

over the variables $\{x, x_1, \ldots, x_k, y, y_1, \ldots, y_k\}$ expresses k-bisimilarity in the sense that $\mathcal{K}, s, \sigma \models \varphi_{\sim}^k$ if and only if $(\sigma(x), \sigma(x_1), \ldots, \sigma(x_k)) \sim^k (\sigma(y), \sigma(y_1), \ldots, \sigma(y_k))$.
To prove this, observe that each conjunct in the fixed point represents one of the clauses of k-bisimilarity in Definition 4.8 (without the clause for nominals as stated above).

Theorem 4.16. H_μ^{2k+2} can express k-bisimilarity for any $k \geq 0$.

4.4 A hierarchy for bounded fragments of $\mathbf{H}_\mu$

In [63] it was shown that the expressive power of the polyadic μ-calculus rises with increasing arity of the formulas. In particular it was shown, that for each level m in the fixed point alternation hierarchy and formulas of arity k – denoted by L_m^k – we have $\mathrm{L}_m^{k+1} \succ \mathrm{L}_m^k$.
The connection between H_μ^k and L_μ^k that we have proven in Section 4.1 suggests that with an increasing number of variables the expressive power of H_μ^k may rise as well. This idea is also strengthened by the previous section which has shown, that while H_μ^k cannot distinguish k-bisimilar states this can be done using at least $2k+2$ variables.
However, both results are not sharp enough to prove that already each additional variable leads to an increase in expressive power, i.e. $\mathrm{H}_\mu^{k+1} \succ \mathrm{H}_\mu^k$. The latter mentioned result only proves that $\mathrm{H}_\mu^{2k+2} \succ \mathrm{H}_\mu^k$ while using the result about the arity hierarchy in the polyadic μ-calculus and the translations from Theorems 4.3 and 4.4 only gives us that $\mathrm{H}_\mu^k(\downarrow, @) \prec \mathrm{H}_\mu^{k+2}(\downarrow, @)$, i.e. the expressive power of the fragments without variable tests rises when adding two variables. To see this, note that the translations from Theorems 4.3 and 4.4 imply that $\mathrm{H}_\mu^k(\downarrow, @) \preceq \mathrm{L}_\mu^{k+1} \preceq \mathrm{H}_\mu^{k+1}(\downarrow, @) \preceq \mathrm{L}_\mu^{k+2} \preceq \mathrm{H}_\mu^{k+2}(\downarrow, @)$ and [63] only implies that $\mathrm{L}_\mu^{k+1} \prec \mathrm{L}_\mu^{k+2}$ for every $k \in \mathbb{N}$.
To obtain the sharper result we employ the invariance under k-bisimulation to prove that there is a formula in H_μ^{k+1} that cannot be expressed by any formula in H_μ^k. It is already known from Lemma 4.12 that no H_μ formula with at most k variables can distinguish the full clique C_{k+1} with $k+1$ states from the full clique C_{k+2} with $k+2$ states. Thus, it suffices to find a formula with $k+1$ variables that can.
The idea for this formula is copied from the winning strategy of Duplicator in the k-bisimulation game on C_{k+2} and C_{k+1}: even if Spoiler marks $k+1$

states in C_{k+2} with his pebbles, then there is always an unmarked state left. Hence, there are at least $k+2$ states.

Lemma 4.17. The formula $\varphi^{k+2} \in \mathrm{H}_\mu^{k+1}$ with

$$\varphi^{k+2} \;:=\; \downarrow x_1.\Diamond\downarrow x_2.\Diamond \dots \Diamond\downarrow x_{k+1}.(\bigwedge_{\substack{i,j=1\\ i\neq j}}^{k} @_{x_i}\neg x_j) \wedge \Diamond(\bigwedge_{i=1}^{k}\neg x_i)$$

is satisfied at a state $s \in S$ if and only if there is a finite path starting at s on which the first $k+2$ nodes are all different.

Corollary 4.18. For every $k \in \mathbb{N}$ we have that $\mathrm{H}_\mu^{k+1} \succ \mathrm{H}_\mu^{k}$.

Proof. Let $k \in \mathbb{N}$ and let $\varphi^{k+2} \in \mathrm{H}_\mu^{k+1}$ be defined as in Lemma 4.17. Obviously, it holds that for every state s in C_{k+2} we have that $C_{k+2}, s \models \varphi^{k+2}$ while on the other hand we also have that $C_{k+1}, t \not\models \varphi^{k+2}$ for all states t in C_{k+1}.
We will show that there is no formula in H_μ^k that is equivalent to φ^{k+2}. For the sake of contradiction, suppose that there exists such a formula φ. By Lemma 4.12 we have that $C_{k+1} \sim^k C_{k+2}$. By Theorem 4.13 we then get that for all states s in C_{k+2} and t in C_{k+1} we have that $C_{k+2}, s \models \varphi$ if and only if $C_{k+1}, t \models \varphi$ which contradicts the assumption that $\varphi \equiv \varphi^{k+2}$. $\square$

To summarise the results of Chapter 4, we have studied the expressive power of H_μ as well as its connection to the polyadic μ-calculus and have shown that each bounded fragment H_μ^k is invariant under k-bisimulation. Using these results we have obtained several relationships between various fragments of H_μ. These relationships are summarised in Figure 4.2.
Using the invariance under k-bisimulations we have shown that the expressive power of the bounded fragments of H_μ grows with each added variable. Furthermore, originating in the study of its relationship with the polyadic μ-calculus, we have obtained that the logics $\mathrm{H}_\mu^k(\downarrow,@)$ lie somewhere between L_μ^k and L_μ^{k+1} in terms of their expressive power. Hence, the unbounded logics L_μ^ω and $\mathrm{H}_\mu(\downarrow,@)$ have the same expressive power.
However, it is still an open problem if one or both of the inclusions in $\mathrm{L}_\mu^k \preceq \mathrm{H}_\mu^k(\downarrow,@) \preceq \mathrm{L}_\mu^{k+1}$ are strict. This also means that the precise relationship between $\mathrm{H}_\mu^k(\downarrow,@)$ and $\mathrm{H}_\mu^{k+1}(\downarrow,@)$ also remains an open problem. Using the results from [63] – also depicted in Figure 4.2 as the inclusions in the bottom line – we can only show that $\mathrm{H}_\mu^k(\downarrow,@) \prec \mathrm{H}_\mu^{k+2}(\downarrow,@)$ but not the sharper result that already one additional variable brings more expressive power as we have shown for the case with added variable tests.

$$\begin{tikzcd}
\mathrm{H}^1_\mu \arrow[r, phantom, "\prec"] \arrow[d, dash, "\rotatebox{90}{$\prec$}" description] & \mathrm{H}^2_\mu \arrow[r, phantom, "\prec"] \arrow[d, dash, "\rotatebox{90}{$\prec$}" description] & \mathrm{H}^3_\mu \arrow[r, phantom, "\prec"] \arrow[d, dash, "\rotatebox{90}{$\prec$}" description] & \cdots \arrow[r, phantom, "\prec"] & \mathrm{H}_\mu \arrow[d, dash, "\rotatebox{90}{$\prec$}" description] \\
\mathrm{H}^1_\mu(\downarrow, @) \arrow[d, dash, "\rotatebox{90}{$\preceq$}" description] \arrow[dr, dash, "\succeq" description, sloped] & \mathrm{H}^2_\mu(\downarrow, @) \arrow[d, dash, "\rotatebox{90}{$\preceq$}" description] \arrow[dr, dash, "\succeq" description, sloped] & \mathrm{H}^3_\mu(\downarrow, @) \arrow[d, dash, "\rotatebox{90}{$\preceq$}" description] \arrow[dr, dash, "\succeq" description, sloped] & \cdots & \mathrm{H}_\mu(\downarrow, @) \arrow[d, dash, "\rotatebox{90}{$\equiv$}" description] \\
\mathrm{L}^1_\mu \arrow[r, phantom, "\prec"] & \mathrm{L}^2_\mu \arrow[r, phantom, "\prec"] & \mathrm{L}^3_\mu \arrow[r, phantom, "\prec"] & \cdots \arrow[r, phantom, "\prec"] & \mathrm{L}^\omega_\mu
\end{tikzcd}$$

Figure 4.2: Expressive Power of the bounded fragments inside of H_μ.

Chapter 5

The Hybrid Branching-Time Hierarchy

We continue to research the expressive power of hybrid branching-time logics. In particular we now study the relative expressive power of all previously introduced hybrid branching-time logics.

We will start by developing Ehrenfeucht-Fraïssé games for the basic hybrid branching-time logic HCTL and show that the defined games capture the expressive power of this logic in the sense that there is no HCTL formula of temporal nesting depth at most n that can distinguish two structures from one another if and only if Duplicator wins the n round game on these two structures. These games will help us in separating HCTL from logics higher up in the syntactic hierarchy.

Then we work towards the main result of this chapter: the hybrid branching-time hierarchy. We present a comparison of all hybrid logics from HCTL up to $\mathrm{HCTL}^*_{\mathsf{pp}}$ as well as the fully hybrid μ-calculus in terms of their expressive power and establish a semantic hierarchy. Both separation results as well as translations between some of the hybrid logics are shown. To achieve these results we develop new model theoretic proof techniques or extend well-known techniques from the area of branching-time model theory.

As it turns out, hybrid logics can make use of certain structural properties to simulate at least some parts of the expressive power of the more expressive branching-time logics and thus the hierarchies on restricted classes of structures differ from the general picture quite a bit depending on the properties of the underlying structures. Possibly one of the most important classes of structures for temporal logics is the class of trees. In the last section of this chapter we will take a close look at this class of structures and try to also fill in the remaining gaps in the hierarchy over trees.

5.1 Ehrenfeucht-Fraïssé Games

Ehrenfeucht-Fraïssé (EF) Games have proven to be a powerful tool in capturing the expressive power of first-order logic [27] as well as some closely related logics.

Such games often prove useful when comparing the expressive power of two logics. They are especially useful to prove that certain properties *cannot* be expressed in the logic for which they are designed because they condense all possibilities of constructing a formula and distinguishing two structures via such a formula into a single comprehensive framework of a two-player game. Hence, the proof that a property is not expressible in a given logic is reduced to finding a winning strategy for one of the players – usually called the Duplicator – in such a game.

For this reason we define Ehrenfeucht-Fraïssé games for hybrid branching-time logics. These hybrid logics extend branching-time logics with certain first-order aspects and thus it is not too surprising that we will find parts of these games that are similar to certain aspects in the FO variants of these games.

Also, similarly to EF games for first-order logic, a single instance of an EF game only captures the expressive power of logical formulas up to a certain degree. In FO it is usually the quantifier depth, i.e. if Duplicator wins the m-round EF game then there is no formula of quantifier depth $\leq m$ that can distinguish these structures. In our case this limiting factor will be the nesting depth of temporal operators.

To show that some property cannot be expressed in a certain logic with the help of EF games it is thus not enough to simply find two structures – one that satisfies the property and one which does not – such that Duplicator has a winning strategy in this game. This only shows that there is no formula up to a certain degree that cannot distinguish these structures. Instead, we need to find two families of structures – one family of structures that satisfies the property and one that does not – such that for each degree $m \in \mathbb{N}$ we can find a pair of structures, one from each family, such that they cannot be distinguished by formulas up to degree m.

EF-Games for HCTL. We begin to define the EF-games for HCTL. For the remainder of this section we fix two Kripke structures $\mathcal{K}_0 = \langle S_0, \rightarrow_0, L_0 \rangle$, $\mathcal{K}_1 = \langle S_1, \rightarrow_1, L_1 \rangle$ and two states $s_0 \in S_0$, $t_0 \in S_1$.

Definition 5.1. The game $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, \mathcal{K}_1, t_0)$ is played between two players – **Spoiler** and **Duplicator** on $\mathcal{K}_0$ and $\mathcal{K}_1$.

The game is played for m rounds and begins with a single pebble placed in each structure on s_0 and t_0. After each round, a new pair of pebbles gets placed in $\mathcal{K}_0$ and $\mathcal{K}_1$. We refer to the pebble placed in $\mathcal{K}_0$ in round n as p_0^n and the pebble placed in $\mathcal{K}_1$ in round n as p_1^n beginning with p_0^0 placed at s_0 and p_1^0 placed at t_0.
Each round n is played according to the following rules: First, **Spoiler** can choose one of the structures $\mathcal{K}_i$, $i \in \{0,1\}$ and a previously placed pebble p_i^j in $\mathcal{K}_i$ for some $j \leq n$. Then he can choose to make one of the following moves:

(X) He chooses a successor of p_i^j and places a new pebble on this successor. **Duplicator** then responds by choosing the pebble p_{1-i}^j in $\mathcal{K}_{1-i}$ and also chooses a successor of p_{1-i}^j on which she places a new pebble.

(U) **Spoiler** chooses a path π_i starting at p_i^j and a position l on this path. Then **Duplicator** chooses a path π_{1-i} starting at p_{1-i}^j and a position l' on this path. Now **Spoiler** has two options.

- Either, he places a new pebble on π_i^l, forcing Duplicator to place her new pebble on $\pi_{1-i}^{l'}$, or
- he chooses $k' < l'$ and places a new pebble on $\pi_{1-i}^{k'}$ on **Duplicator**'s path. Afterwards, **Duplicator** can choose $k < l$ and places a new pebble on π_i^k on **Spoiler**'s path.

(G) He chooses a path π_i starting at p_i^j. Then **Duplicator** chooses a path π_{1-i} starting at p_{1-i}^j. Now **Spoiler** chooses $l \in \mathbb{N}$ and places a new pebble on π_{1-i}^l and after that **Duplicator** can choose some $l' \in \mathbb{N}$ and place a new pebble on $\pi_i^{l'}$.

It is Duplicator's task to maintain the following conditions after each round i:

- For all pairs of pebbles (p_0^j, p_1^j) it holds that the states marked by these pebbles agree on all atomic propositions $p \in \mathit{Prop}$ and all nominals $n \in \mathit{Nom}$.
- For all pairs of pebbles (p_0^j, p_1^j) and (p_0^k, p_1^k) with $0 \leq j, k \leq i$ it holds that $p_0^j = p_0^k$ if and only if $p_1^j = p_1^k$.

Spoiler wins if Duplicator cannot maintain these conditions after some round. Duplicator wins if she can survive m rounds.

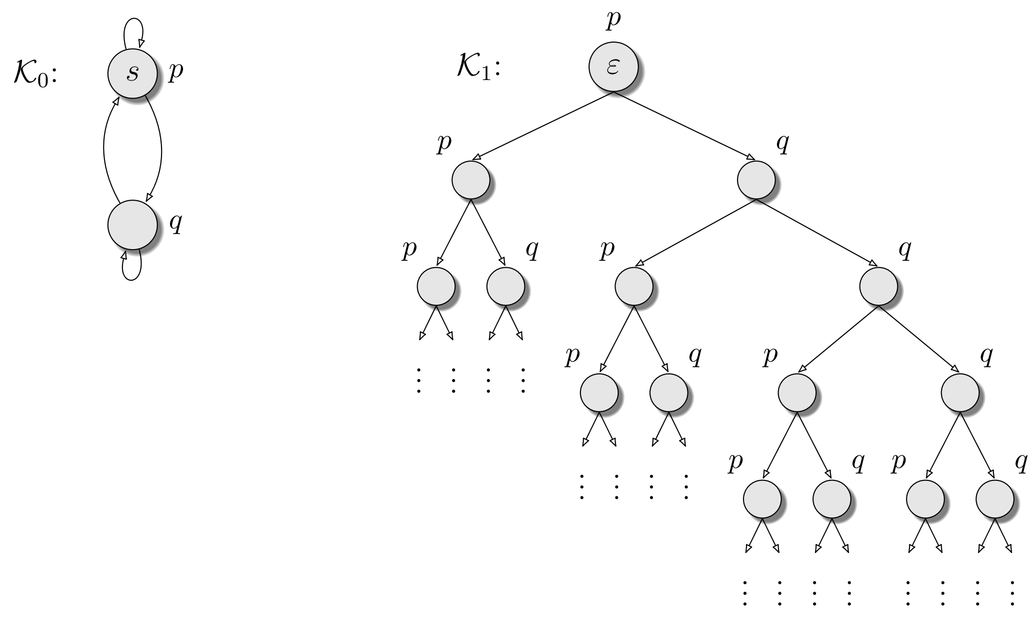


Figure 5.1: A Kripke structure $\mathcal{K}_0$ and its tree unraveling $\mathcal{K}_1$.

Note that the second and third move at first glance seem to be quite similar. However, the third option uses unbounded paths while the second only uses finite parts of a path but forces Duplicator to make the choice of where to end her finite path earlier and thus gives more choices to Spoiler in the second part of this move.
To illustrate these games we give a few examples.

Example 5.2. First, take a look at the two structures $\mathcal{K}_0, \mathcal{K}_1$ depicted in Figure 5.1. These depict a simple Kripke structure $\mathcal{K}_0$ that consists of only two states and its tree unraveling $\mathcal{K}_1$. As we know, a Kripke structure and its tree unraveling are bisimilar and thus no CTL, CTL* or even L$_\mu$ formula can distinguish between these two structures.
However, it is quite easy to see that Spoiler has a winning strategy already in $\mathcal{G}^1_{\text{HCTL}}(\mathcal{K}_0, s, \mathcal{K}_1, \varepsilon)$. For this he simply makes a single (X) move along the edge in $\mathcal{K}_1$ to the child that is also labelled with p. Duplicator is then forced to move in $\mathcal{K}_0$. She can either move to the state labelled q and lose because the atomic propositions do not match or she can stay at s and lose because we have that $p^0_1 \neq p^1_1$ but also $p^0_0 = p^1_0$.
It is also easy to see that the HCTL formula $p \wedge {\downarrow} x.\mathsf{EX}(p \wedge \neg x)$, which mimics this strategy, distinguishes both structures.

Example 5.3. For the next example take a look at the structures $\mathcal{K}_2$ and

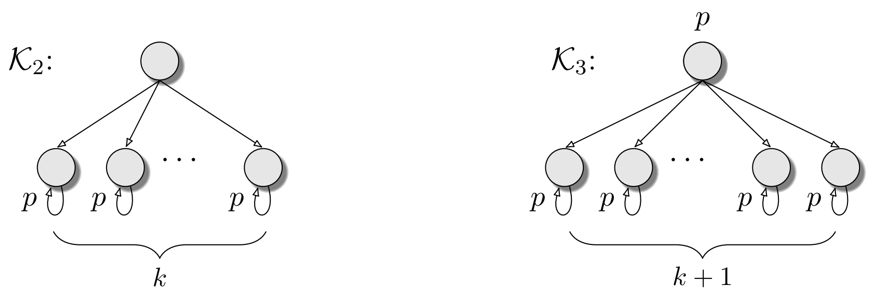


Figure 5.2: Two Kripke structures.

$\mathcal{K}_3$ depicted in Figure 5.2.
Both structures only differ in the number of their successors. $\mathcal{K}_2$ has k successors and $\mathcal{K}_3$ has $k+1$ successors for some $k \in \mathbb{N}$. It is easy to see that Spoiler wins the EF game starting at the roots of both trees if the game goes on for at least $k+1$ rounds by starting each move at the root and simply marking another successor in each round. In round $k+1$ he then moves in $\mathcal{K}_3$ to the last non-marked successor. Duplicator cannot mimic this anymore since all states in $\mathcal{K}_2$ have already been marked.
Similarly Duplicator wins if the game only lasts for k or less rounds. If Spoiler picks a successor (or a path through some successor) that is not marked then Duplicator does so as well in the other structure and if Spoiler picks an already marked successor then Duplicator picks the correspondingly marked successor in the other structure. The moves (U) and (G) can be treated along the same lines since they are essentially the same as the (X) move in these two structures.
This strategy is similar to the one used in Lemma 4.12 to show that the clique with $k+1$ and the clique with more than $k+1$ states are k-bisimilar.
An HCTL formula that distinguishes both structures can also easily be derived from Spoiler's winning strategy in the $(k+1)$-round game:

$$\downarrow \mathsf{root}.\mathsf{EX}\downarrow x_1.\ @_{\mathsf{root}}\,\mathsf{EX}\big(\neg x_1 \wedge \downarrow x_2.\ @_{\mathsf{root}} \ldots \downarrow x_k.\ @_{\mathsf{root}}\,\mathsf{EX}(\bigwedge_{i=1}^{k} \neg x_i) \ldots\big).$$

The formula again mimics Spoiler's strategy in naming each successor and requiring that they are all different.

Another – slightly more involved – example that also uses the other available moves aside from (X) will be presented in Theorem 5.16 to show that certain HCTL$^*_{\mathsf{ss}}$ properties cannot be expressed in HCTL.

We now prove that these games capture the expressive power of HCTL formulas, i.e. there is a formula in HCTL with nesting depth up to $m \in \mathbb{N}$ that can distinguish the two structures in question if and only if Spoiler wins the m-round EF game on these structures. To prove this we first need a normal form for HCTL formulas.

Lemma 5.4. Let $\varphi \in$ HCTL. Then there is a formula $\varphi' \in$ HCTL with $\varphi \equiv \varphi'$ such that $\mathsf{nd}(\varphi') = \mathsf{nd}(\varphi)$ and φ' is built using only temporal operators EX, EU and EG.

Proof. The proof of this Lemma is simply by substituting the other temporal operators AX, AU via the following equivalences:

$$\begin{aligned} \mathsf{AX}\varphi &\equiv \neg\mathsf{EX}\neg\varphi, \\ \mathsf{A}\varphi_1\mathsf{U}\varphi_2 &\equiv \neg\mathsf{EG}\neg\varphi_2 \wedge \neg\mathsf{E}(\neg\varphi_2\mathsf{U}(\neg\varphi_1 \wedge \neg\varphi_2)). \end{aligned}$$

Note that both equivalences maintain the temporal nesting depth. □

Using this normal form for HCTL formulas we can prove the first direction.

Theorem 5.5. If Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, \mathcal{K}_1, t_0)$ then for all formulas $\varphi \in$ HCTL with $\mathsf{nd}(\varphi) \leq m$ it holds that $\mathcal{K}_0, s_0 \models \varphi$ if and only if $\mathcal{K}_1, t_0 \models \varphi$.

Proof. We show an extended statement that accounts for pebbles that have already been placed and free variables in the formula:
Let $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \ldots, s_{n-1}, \mathcal{K}_1, t_0, t_1, \ldots, t_{n-1})$ be the game where n pebbles have already been placed and let $\sigma : \mathit{Var} \to S_0$, $\sigma' : \mathit{Var} \to S_1$ such that for all $x \in \mathit{Var}$ and $0 \leq i < n$ it holds that $\sigma(x) = s_i$ if and only if $\sigma'(x) = t_i$ and especially $\sigma(x_j) = s_j$, resp. $\sigma'(x_j) = t_j$ for $0 \leq j < n$. The variables $x_0, \ldots, x_{n-1}$ will be used as free variables in the formula.
We show that if Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \ldots, s_{n-1}, \mathcal{K}_1, t_0, t_1, \ldots, t_{n-1})$ then it holds for all formulas $\varphi \in$ HCTL with free variables $x_0, \ldots, x_{n-1}$ and $\mathsf{nd}(\varphi) \leq m$ that for all $0 \leq i < n$ we have $\mathcal{K}_0, s_i, \sigma \models \varphi$ if and only if $\mathcal{K}_1, t_i, \sigma' \models \varphi$.
It should be clear that the statement of the theorem follows from the above claim for closed formulas $\varphi \in$ HCTL. Moreover, by Lemma 5.4 it suffices to look at formulas built with only temporal operators EX, EU and EG.
We show this extended statement by an induction on the number of rounds m and only show the moves where **Spoiler** decides to move on $\mathcal{K}_0$. The other cases are completely symmetrical.
So, let $m = 0$ and suppose that Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \ldots, s_{n-1}, \mathcal{K}_1, t_0, t_1, \ldots, t_{n-1})$. We show the statement by a separate induction on formulas φ of nesting depth at most 0.

- By the first part of the winning conditions for Duplicator it follows that for all $\varphi = p$ or $\varphi = n$ and all $0 \leq i < n$ it holds that $\mathcal{K}_0, s_i, \sigma \models \varphi$ if and only if $\mathcal{K}_1, t_i, \sigma' \models \varphi$.

- The case for $\varphi = x_j$ for one of the free variables $x_0, \dots, x_{n-1} \in \mathit{Var}$ follows by construction of σ and σ'.

- The cases for boolean connectives follow with simple semantical arguments as usual.

- Suppose that $\varphi = {\downarrow}x.\chi$ and $\mathcal{K}_0, s_i, \sigma \models \varphi$ for some $0 \leq i < n$. Then $\mathcal{K}_0, s_i, \sigma[x \mapsto s_i] \models \chi$. By assumption, $x \notin \{x_0, \dots, x_{n-1}\}$ (otherwise $x_0, \dots, x_{n-1}$ would not be free). Note that, since σ and σ' satisfy the condition that $\sigma(x) = s_i$ iff $\sigma'(x) = t_i$ and $\sigma(x_i) = s_i$ resp. $\sigma'(x_i) = t_i$ for all i, so do $\sigma[x \mapsto s_i]$ and $\sigma'[x \mapsto t_i]$. Furthermore, since $\sigma(x_i) = s_i$ we also have that $\mathcal{K}_0, s_i, \sigma[x \mapsto s_i] \models \chi[x_i/x]$ where all free occurrences of x have been replaced by x_i because x and x_i are bound to the same state s_i anyways. The formula $\chi[x_i/x]$ only has free variables $x_0, \dots, x_{n-1}$ and thus we can use the induction hypothesis to obtain that $\mathcal{K}_1, t_i, \sigma'[x \mapsto t_i] \models \chi[x_i/x]$. With the same argument as before we then also get that $\mathcal{K}_1, t_i, \sigma'[x \mapsto t_i] \models \chi$ and thus also $\mathcal{K}_1, t_i, \sigma' \models {\downarrow}x.\chi$.

- Lastly, suppose that $\varphi = @_{x_j} \chi$ for $0 \leq j < n$ and $\mathcal{K}_0, s_i, \sigma \models \varphi$. Remember, that φ only has free variables $x_0, \dots, x_{n-1}$, thus jumps can only be to one of those variables. We get that $\mathcal{K}_0, \sigma(x_j), \sigma \models \chi$ and with the assumption that $\sigma(x_j) = s_j$ we have that $\mathcal{K}_0, s_j, \sigma \models \chi$. By the induction hypothesis for χ we get that $\mathcal{K}_1, t_j, \sigma' \models \chi$ and again with $t_j = \sigma'(x_j)$ we get that $\mathcal{K}_1, t_i, \sigma' \models @_{x_j} \chi$.

This concludes the base case for $m = 0$.
So, let $m \geq 1$ and suppose that the statement already holds for the $m - 1$ round game. Suppose again that Duplicator wins $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \dots, s_{n-1}, \mathcal{K}_1, t_0, t_1, \dots, t_{n-1})$ and that $\mathsf{nd}(\varphi) \leq m$. We show that $\mathcal{K}_0, s_i, \sigma \models \varphi$ if and only if $\mathcal{K}_1, t_i, \sigma' \models \varphi$ by an induction over the structure of φ. Without loss of generality we can assume that φ only uses temporal operators $\mathsf{EX}, \mathsf{EU}, \mathsf{EG}$. The cases for atomic formulas, boolean connectives and hybrid operators can be shown in the same way as for $m = 0$ with a separate induction over the structure of HCTL formulas φ with $\mathsf{nd}(\varphi) = m$. We show the three remaining cases for $\varphi = \mathsf{EX}\chi$, $\varphi = \mathsf{E}\chi_1\mathsf{U}\chi_2$ and $\varphi = \mathsf{EG}\chi$.

- Suppose $\varphi = \mathsf{EX}\chi$ and $\mathcal{K}_0, s_i, \sigma \models \varphi$ for some $0 \leq i < n$. Then there is a successor s'_i of s_i in $\mathcal{K}_0$ such that $\mathcal{K}_0, s'_i, \sigma \models \chi$. Suppose that Spoiler chooses to make a successor move on $\mathcal{K}_0$ and chooses to move from s_i to s'_i. Since **Duplicator** wins $\mathcal{G}^m_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \dots, s_{n-1}, \mathcal{K}_1, t_0, t_1, \dots, t_{n-1})$, she can choose a successor t'_i of t_i such that she wins $\mathcal{G}^{m-1}_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \dots, s_{n-1}, s'_i, \mathcal{K}_1, t_0, t_1, \dots, t_{n-1}, t'_i)$. Since $\mathcal{K}_0, s'_i, \sigma \models \chi$ and $\mathsf{nd}(\chi) < m$ we can use the induction hypothesis to conclude that $\mathcal{K}_1, t'_i, \sigma' \models \chi$[1]. And since t'_i is a successor of t_i we get that $\mathcal{K}_1, t_i, \sigma' \models \mathsf{EX}\chi$.

- Suppose $\varphi = \mathsf{E}\chi_1\mathsf{U}\chi_2$ and $\mathcal{K}_0, s_i, \sigma \models \varphi$ for some $0 \leq i < n$. Then there is a path π_0 starting at s_i and some l such that $\mathcal{K}_0, \pi_0^l, \sigma \models \chi_2$ and for all $j < l$, $\mathcal{K}_0, \pi_0^j, \sigma \models \chi_1$. Suppose **Spoiler** chooses to play π_0 and l on $\mathcal{K}_0$. Since **Duplicator** wins, her winning strategy tells her to choose some path π_1 and l' starting at t_i.

  Now **Spoiler** has two choices:

  - First, he can choose to place a new pebble on π_0^l which forces **Duplicator** to place one on $\pi_1^{l'}$. Since **Duplicator** is playing a winning strategy this means she wins from $\mathcal{G}^{m-1}_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \dots, s_{n-1}, \pi_0^l, \mathcal{K}_1, t_0, t_1, \dots, t_{n-1}, \pi_1^{l'})$. Then, since $\mathcal{K}_0, \pi_0^l, \sigma \models \chi_2$ and $\mathsf{nd}(\chi_2) < m$, we can use the induction hypothesis and deduce that $\mathcal{K}_1, \pi_1^{l'}, \sigma' \models \chi_2$.
  - Secondly, suppose that **Spoiler** plays some $k' < l'$ and places a new pebble at $\pi_1^{k'}$. Since **Duplicator** is winning, she can choose some $k < l$ and place her pebble on π_0^k such that she wins $\mathcal{G}^{m-1}_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \dots, s_{n-1}, \pi_0^k, \mathcal{K}_1, t_0, t_1, \dots, t_{n-1}, \pi_1^{k'})$. Now, since $\mathcal{K}_0, \pi_0^k, \sigma \models \chi_1$ and $\mathsf{nd}(\chi_1) < m$ we can deduce that $\mathcal{K}_1, \pi_1^{k'}, \sigma' \models \chi_1$. In fact, this is true for every $k' < l'$ since it was **Spoiler**'s choice.

  Both cases together give us that there is some l' on π_1 such that $\mathcal{K}, \pi_1^{l'}, \sigma' \models \chi_2$ and for all $k' < l'$ we have that $\mathcal{K}, \pi_1^{k'}, \sigma' \models \chi_1$. Hence, we get that $\mathcal{K}_1, t_i, \sigma' \models \mathsf{E}\chi_1\mathsf{U}\chi_2$.

- Lastly, suppose that $\varphi = \mathsf{EG}\chi$ and that $\mathcal{K}_0, s_i, \sigma \models \varphi$ for some $0 \leq i < n$. Then there is a path π_0 starting at s_i such that for all $i \in \mathbb{N}$ it holds that $\mathcal{K}_0, \pi_0^i, \sigma \models \chi$. Suppose that **Spoiler** plays π_0. Let π_1 be **Duplicator**'s answer according to her winning strategy. Suppose further that **Spoiler** decides to play some $l \in \mathbb{N}$ and **Duplicator**'s

[1]To be precise we would really need new variable assignments that account for the fact that $\sigma(x_n) = s'_i$ and $\sigma'(x_n) = t'_i$ so that we can use the induction hypothesis properly. However, due to the fact that x_n is not a free variable in φ, the satisfaction of φ does not depend on the assignment of x_n and this is just a minor technicality. The same is true for the assumptions in the case $\varphi = \mathsf{E}\chi_1\mathsf{U}\chi_2$.

winning strategy tells her to respond with $l' \in \mathbb{N}$. Then **Duplicator** wins $\mathcal{G}^{m-1}_{\text{HCTL}}(\mathcal{K}_0, s_0, s_1, \ldots, s_{n-1}, \pi_0^{l'}, \mathcal{K}_1, t_0, t_1, \ldots, t_{n-1}, \pi_1^{l})$ and it holds that $\mathcal{K}_0, \pi_0^{l'}, \sigma \models \chi$. By the induction assumption we thus have that $\mathcal{K}_1, \pi_1^{l}, \sigma' \models \chi$. As before this holds for every $l \in \mathbb{N}$ since it was **Spoiler**'s choice. Thus we have that $\mathcal{K}_1, t_i, \sigma' \models \varphi$.

The reverse direction, i.e. if $\mathcal{K}_1, t_i, \sigma' \models \varphi$ then also $\mathcal{K}_0, s_i, \sigma \models \varphi$, can be shown in the same way. This concludes the proof. □

The other direction, i.e. if Spoiler wins then there is a formula that distinguishes both structures, holds as well – at least for structures with finite branching-degree. But in order to prove this direction we first need some technical details.

Definition 5.6. Let $m \in \mathbb{N}$, $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$ be a finite set of atomic propositions and $\{x_1, \ldots, x_n\} \subseteq \mathit{Var}$ be a finite subset of variables.
A formula $\varphi \in \text{HCTL}$ with $\mathsf{nd}(\varphi) = m$ over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_1, \ldots, x_n\}$ is in *hybrid conjunctive normal form of degree m* – or short hybrid CNF of degree m – if it is of the form

$$\Big(\bigwedge_{i \in I} \bigvee_{j \in J_i} \mathsf{Q}_{i,j}.l_{i,j}\Big),$$

where all $l_{i,j}$ are either of the form p, x, $\mathsf{EX}\chi$, $\mathsf{E}\chi_1\mathsf{U}\chi_2$, $\mathsf{AX}\chi$ or $\mathsf{A}\chi_1\mathsf{U}\chi_2$ with $p \in \mathit{Prop}_{\mathsf{fin}}$, $x \in \{x_1, \ldots, x_n\}$ and $\chi, \chi_1, \chi_2 \in \text{HCTL}$ or negations thereof and $\mathsf{Q}_{i,j}$ is a sequence of the hybrid operators $\downarrow$ and $@$ over the variables $\{x_1, \ldots, x_n\}$.

Lemma 5.7. For each $\varphi \in \text{HCTL}$ and $\mathsf{nd}(\varphi) = m \in \mathbb{N}$, there is a formula $\varphi' \in \text{HCTL}$ in hybrid conjunctive normal form of degree m such that $\varphi \equiv \varphi'$.

Proof. We shortly describe how to transform a formula $\varphi \in \text{HCTL}$ with $\mathsf{nd}(\varphi) = 0$. The transformation for higher temporal nesting depth is essentially the same but only on the top level, above the outermost path quantifiers.
So assume that $\varphi \in \text{HCTL}$ with temporal nesting depth 0. We start with the usual propositional equivalences, as well as the equivalences $\downarrow x.(\varphi_1 \otimes \varphi_2) \equiv \downarrow x.\varphi_1 \otimes \downarrow x.\varphi_2$, $@_x(\varphi_1 \otimes \varphi_2) \equiv @_x \varphi_1 \otimes @_x \varphi_2$ with $\otimes \in \{\wedge, \vee\}$, $\neg \downarrow x.\varphi \equiv \downarrow x.\neg\varphi$ and $\neg @_x \varphi \equiv @_x \neg\varphi$ to push negations, binders and jumps inwards until they only occur right in front of atomic propositions.
After this we use the usual distributive laws to achieve a conjunctive normal form above the hybrid operators. □

Lemma 5.8. Let $m \in \mathbb{N}$, $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$ be a finite set of atomic propositions and $\{x_1, \dots, x_n\} \subseteq \mathit{Var}$ be a finite subset of variables. Then there are only finitely many non-equivalent formulas $\varphi \in$ HCTL over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_1, \dots, x_n\}$ with $\mathsf{nd}(\varphi) \leq m$.

Proof. By Lemma 5.7 it suffices to argue that there are only finitely many non-equivalent formulas in hybrid CNF of degree m. So, let $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_1, \dots, x_n\}$ be fixed.
We argue the case for $m = 0$. Formulas $\varphi \in$ HCTL in hybrid CNF of degree 0 are built using only atomic propositions and variable tests (and their negations) as literals. Thus, for fixed and finite $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$ and $\{x_1, \dots, x_n\} \subseteq \mathit{Var}$ there are only finitely many literals available.
Next is the sequence of binders and jumps. Note that there are no other temporal or boolean constructs in between. Thus, the variables $x_1, \dots, x_n$ represent at most n different states and by only jumping and rebinding the variables at states that already are represented by a variable there are at most n^n possibilities to rearrange these states – most certainly not all of them are possible using only binders and jumps – and another n possibilities at which of the n states the formula continues before evaluating the literal underneath.
Combining both aspects, we obtain that there are at most $c := (|\mathit{Prop}_{\mathsf{fin}}| + n) \cdot n^n \cdot n$ many different "atomic" formulas at the lowest level of such a hybrid CNF of degree 0. Hence, there are at most 2^c many possible clauses which can be built using these and thus only 2^{2^c} many different hybrid CNFs of degree 0. This proves the case for $m = 0$.
For $m \geq 1$ the argumentation is similar. Note that there are additional possible literals underneath the sequences of binders and jumps built by the constructions $\mathsf{EX}\chi$, $\mathsf{E}\chi_1\mathsf{U}\chi_2$, $\mathsf{AX}\chi$ or $\mathsf{A}\chi_1\mathsf{U}\chi_2$ with "smaller" subformulas χ, χ_1, χ_2 and negations thereof. By assumption however, there are only finitely many non-equivalent formulas χ, χ_1, χ_2 and thus also only finitely many additional literals that need to be considered. The remaining argument is the same as for $m = 0$. $\square$

Definition 5.9. Let $\mathcal{K} = \langle S, \rightarrow, L \rangle$ be a Kripke structure, $s \in S$, $\sigma : \mathit{Var} \rightarrow S$ be a variable assignment and $\{x_0, \dots, x_n\} \subseteq \mathit{Var}$ be a finite subset of variables and $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$ a finite subset of atomic propositions. We define the *characteristic formula of s and σ up to degree m over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_n\}$* as

$$\chi^{m,n}_{s,\sigma} := \bigwedge_{\substack{\varphi \in \mathrm{HCTL}(x_0,\dots,x_n) \\ \mathsf{nd}(\varphi) \leq m \\ s,\sigma \models \varphi}} \varphi \wedge \bigwedge_{\substack{\varphi \in \mathrm{HCTL}(x_0,\dots,x_n) \\ \mathsf{nd}(\varphi) \leq m \\ s,\sigma \not\models \varphi}} \neg\varphi,$$

where $\mathrm{HCTL}(x_0, \dots, x_n)$ means all HCTL formulas over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_n\}$ including formulas with potentially free variables from $\{x_0, \dots, x_n\}$.

The characteristic formula gathers all facts up to temporal depth m that hold at a state given a certain variable assignment. Since by Lemma 5.8, there are only finitely many pairwise non-equivalent such formulas this is indeed a proper formula. Also, notice that $\mathsf{nd}(\chi^{m,n}_{s,\sigma}) \leq m$.

Lemma 5.10. Let $\mathcal{K}_0, \mathcal{K}_1$ be two Kripke structures over a finite set of atomic propositions $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$, $s \in S_0$, $t \in S_1$, $\sigma_0 : \mathit{Var} \to S_0$ and $\sigma_1 : \mathit{Var} \to S_1$.
If $\mathcal{K}_0, s, \sigma_0 \models \chi^{m,n}_{s,\sigma_0}$ and $\mathcal{K}_1, t, \sigma_1 \models \chi^{m,n}_{s,\sigma_0}$ then it holds for all $\varphi \in$ HCTL over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_n\}$ (including formulas with potential free variables from $\{x_0, \dots, x_n\}$) with $\mathsf{nd}(\varphi) \leq m$ that $\mathcal{K}_0, \sigma_0(x_i), \sigma_0 \models \varphi$ if and only if $\mathcal{K}_1, \sigma_1(x_i), \sigma_1 \models \varphi$ for all $0 \leq i \leq n$.

Proof. Suppose the contrary for the sake of contradiction. Thus, there is a formula $\varphi \in$ HCTL over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_n\}$ with $\mathsf{nd}(\varphi) \leq m$ and $0 \leq i \leq n$ such that $\mathcal{K}_0, \sigma_0(x_i), \sigma_0 \models \varphi$ but $\mathcal{K}_1, \sigma_1(x_i), \sigma_1 \not\models \varphi$ (or vice versa) and further we have that $\mathcal{K}_0, s, \sigma_0 \models \chi^{m,n}_{s,\sigma_0}$ and $\mathcal{K}_1, t, \sigma_1 \models \chi^{m,n}_{s,\sigma_0}$.
By the semantics of HCTL we then have that $\mathcal{K}_0, s, \sigma_0 \models @_{x_i} \varphi$ but $\mathcal{K}_1, t, \sigma_1 \not\models @_{x_i} \varphi$. However, since $\mathsf{nd}(@_{x_i} \varphi) \leq m$ we also know that it must be equivalent to a conjunct in $\chi^{m,n}_{s,\sigma_0}$ and consequently we have that $\mathcal{K}_1, t, \sigma_1 \not\models \chi^{m,n}_{s,\sigma_0}$ which contradicts the assumption. $\square$

With this we have all the necessary tools together to prove the other direction.

Theorem 5.11. Let $\mathcal{K}_0, \mathcal{K}_1$ be two Kripke structures over a finite set of atomic propositions $\mathit{Prop}_{\mathsf{fin}} \subseteq \mathit{Prop}$ and with finite branching-degree. If it holds for all formulas $\varphi \in$ HCTL with $\mathsf{nd}(\varphi) \leq m$ that $\mathcal{K}_0, s_0 \models \varphi$ if and only if $\mathcal{K}_1, t_0 \models \varphi$ then **Duplicator** wins $\mathcal{G}^m_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \mathcal{K}_1, t_0)$.

Proof. We prove a stronger statement that also accounts for free variables that may occur throughout the game:
If it holds for all formulas $\varphi \in$ HCTL over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_{n-1}\}$ with potential free variables from $\{x_0, \dots, x_{n-1}\}$ and $\mathsf{nd}(\varphi) \leq m$ as well as all variable assignments $\sigma_0 : \mathit{Var} \to S_0$, $\sigma_1 : \mathit{Var} \to S_1$ with $\sigma_0(x_i) = s_i$, $\sigma_1(x_i) = t_i$ for $0 \leq i \leq n-1$ that $\mathcal{K}_0, s_0, \sigma_0 \models \varphi$ if and only if $\mathcal{K}_1, t_0, \sigma_1 \models \varphi$ then **Duplicator** wins $\mathcal{G}^m_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \dots, s_{n-1}, \mathcal{K}_1, t_0, \dots, t_{n-1})$.
We will prove the statement by an induction over m.
To begin the proof by induction, suppose that $m = 0$ and that it holds for all formulas $\varphi \in$ HCTL over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_{n-1}\}$ with potentially free

variables from $\{x_0, \dots, x_{n-1}\}$ and all variable assignments $\sigma_0 : \mathit{Var} \to S_0$, $\sigma_1 : \mathit{Var} \to S_1$ with $\sigma_0(x_i) = s_i$, $\sigma_1(x_i) = t_i$ for $1 \leq i \leq n-1$ that

$$\mathcal{K}_0, s_0, \sigma_0 \models \varphi \text{ if and only if } \mathcal{K}_1, t_0, \sigma_1 \models \varphi.$$

Then it especially holds that $\mathcal{K}_0, s_0, \sigma_0 \models \bigwedge_{i=0}^{n-1} @_{x_i} \chi^{0,n-1}_{s_i,\sigma_0}$ and $\mathcal{K}_1, t_0, \sigma_1 \models \bigwedge_{i=0}^{n-1} @_{x_i} \chi^{0,n-1}_{s_i,\sigma_0}$.
We claim that Duplicator wins $\mathcal{G}^m_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \dots, s_{n-1}, \mathcal{K}_1, t_0, \dots, t_{n-1})$. Suppose for the sake of contradiction that this is not the case. Then by the winning conditions of these games one of Duplicator's winning conditions must be violated. In the first case, suppose some $p \in \mathit{Prop}$ or $n \in \mathit{Nom}$ holds at some s_i but not at t_i (or vice versa), so $\mathcal{K}_1, t_0, \sigma_1 \not\models @_{x_i} p$ or $\mathcal{K}_1, t_0, \sigma_1 \not\models @_{x_i} n$ while $\mathcal{K}_1, s_0, \sigma_1 \models @_{x_i} p$ or $\mathcal{K}_1, s_0, \sigma_1 \models @_{x_i} n$. This contradicts the assumption. The second case is similar. Suppose $s_i = s_j$ but not $t_i = t_j$ for some $0 \leq i, j \leq n-1$. Then $\mathcal{K}_0, s_0, \sigma_0 \models @_{x_i} x_j$ but $\mathcal{K}_1, t_0, \sigma_1 \not\models @_{x_i} x_j$ which again is a contradiction. Thus Duplicator wins $\mathcal{G}^0_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \dots, s_{n-1}, \mathcal{K}_1, t_0, \dots, t_{n-1})$ which concludes the base case.
So, suppose $m \geq 1$ and that it holds for all formulas $\varphi \in \mathrm{HCTL}$ over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_{n-1}\}$ with potentially free variables from $\{x_0, \dots, x_{n-1}\}$ and all variable assignments $\sigma_0 : \mathit{Var} \to S_0$, $\sigma_1 : \mathit{Var} \to S_1$ with $\sigma_0(x_i) = s_i$, $\sigma_1(x_i) = t_i$ for $1 \leq i \leq n-1$ that $\mathcal{K}_0, s_0, \sigma_0 \models \varphi$ if and only if $\mathcal{K}_1, t_0, \sigma_1 \models \varphi$. We construct a winning strategy for Duplicator in $\mathcal{G}^m_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \dots, s_{n-1}, \mathcal{K}_1, t_0, \dots, t_{n-1})$ depending on which move Spoiler makes.
Suppose first that **Spoiler** chooses to make an (X)-move and suppose that he chooses to move from some s_i to a successor s'_i (the case that Spoiler moves on $\mathcal{K}_1$ is completely symmetrical). Let $\chi^{m-1,n}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$ be the characteristic formula of s'_i and $\sigma_0[x_n \mapsto s'_i]$ up to degree $m-1$. Obviously we have that $\mathcal{K}_0, s'_i, \sigma_0[x_n \mapsto s'_i] \models \chi^{m-1,n}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$.
We claim that there exists a successor t'_i of t_i on $\mathcal{K}_1$ such that $\mathcal{K}_1, t'_i, \sigma_1[x_n \mapsto t'_i] \models \chi^{m-1,n}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$. Suppose this was not the case, then we have that $\mathcal{K}_1, t_0, \sigma_1 \not\models @_{x_i} \mathsf{EX} {\downarrow} x_n . \chi^{m-1,n}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$ while clearly by construction we have $\mathcal{K}_0, s_0, \sigma_0 \models @_{x_i} \mathsf{EX} {\downarrow} x_n . \chi^{m-1,n}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$. And since the formula has a temporal nesting depth of at most m this contradicts the assumption that (s_0, σ_0) and (t_0, σ_1) agree on all formulas of temporal nesting depth at most m.
Thus, **Duplicator** can choose to answer Spoiler's (X)-move with t'_i. It remains to be shown that this is a winning strategy for Duplicator.
For this, observe that $\mathcal{K}_1, t'_i, \sigma_1[x_n \mapsto t'_i] \models \chi^{m-1}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$ and $\mathcal{K}_0, s'_i, \sigma_0[x_n \mapsto t'_i] \models \chi^{m-1}_{s'_i,\sigma_0[x_n \mapsto s'_i]}$. Further, we have that $\sigma_0(x_0) = s_0$ and $\sigma_1(x_0) = t_0$. Thus with Lemma 5.10 we get that it holds for all $\varphi \in \mathrm{HCTL}$ over $\mathit{Prop}_{\mathsf{fin}}$ and $\{x_0, \dots, x_n\}$ with potential free variables from $\{x_0, \dots, x_n\}$ and $\mathsf{nd}(\varphi) \leq$

$m-1$ that $\mathcal{K}_0, s_0, \sigma_0 \models \varphi$ if and only if $\mathcal{K}_1, t_0, \sigma_1 \models \varphi$. And with the induction hypothesis we thus get that **Duplicator** wins $\mathcal{G}^{m-1}_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \ldots, s_{n-1}, s'_i, \mathcal{K}_1, t_0, \ldots, t_{n-1}, t'_i)$.

Suppose now that **Spoiler** makes a (G)-move and that he chooses a path π starting from some s_i. Let $\chi^{m-1,n}_{\pi^j,\sigma_0[x_n \mapsto \pi^j]}$, $j \geq 0$ be the characteristic formulas of π^j and $\sigma_0[x_n \mapsto \pi^j]$ up to degree $m-1$. It should be clear by construction that $\mathcal{K}_0, \pi^j, \sigma_0[x_n \mapsto \pi^j] \models \chi^{m-1,n}_{\pi^j,\sigma_0[x_n \mapsto \pi^j]}$ for every $j \geq 0$. Let $\psi := \bigvee_{j \geq 0} {\downarrow} x_n.\chi^{m-1,n}_{\pi^j,\sigma_0[x_n \mapsto \pi^j]}$. By Lemma 5.8, ψ can be considered as finite (or at least as equivalent to a finite formula).

Consider the tree structure $\mathcal{T}_1$ of all finite paths $\tau_1 \ldots \tau_n$ starting at t_i such that $\mathcal{K}_1, \tau_i, \sigma_1 \models \psi$ for all $0 \leq i \leq n$ where finite paths are connected via the direct prefix relation, i.e. $(\tau_1, \ldots, \tau_n) \to (\tau_1, \ldots, \tau_{n+1})$. By assumption, $\mathcal{K}_1$ has finite branching-degree and thus $\mathcal{T}_1$, which is a thinned out subset of the tree unfolding of $\mathcal{K}_1$, has also finite branching-degree.

We claim that $\mathcal{T}_1$ has arbitrarily long finite paths. Suppose for the sake of contradiction the opposite. Then there would be a distance m from the root, such that for all paths τ starting at t_i we have that $\mathcal{K}_1, \tau_j, \sigma_1 \not\models \psi$ for some $j \leq m$. However, this means that $\mathcal{K}_1, t_i, \sigma_1 \not\models \mathsf{EG}\psi$ which is clearly satisfied by construction at s_i on the path π. And since $\mathsf{nd}(\mathsf{EG}\psi) \leq m$ this contradicts the assumption.

Using König's Lemma we can conclude that there is an infinite path τ starting at t_i such that $\mathcal{K}_1, \tau_i, \sigma_1 \models \psi$ for every $i \geq 0$. We construct **Duplicator**'s strategy to respond with τ to **Spoiler**'s move.

Let $l' \in \mathbb{N}$ be **Spoiler**'s choice on τ. Since $\mathcal{K}_1, \tau_{l'}, \sigma_1 \models \psi$ there has to be some $l \geq 0$ such that $\mathcal{K}_1, \tau_{l'}, \sigma_1 \models {\downarrow} x_n.\chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$. Let **Duplicator** choose $l \in \mathbb{N}$.

Again, it remains to be shown that **Duplicator** wins $\mathcal{G}^{m-1}_{\mathrm{HCTL}}(\mathcal{K}_0, s_0, \ldots, s_{n-1}, \pi^l, \mathcal{K}_1, t_0, \ldots, t_{n-1}, \tau_{l'})$. For this, observe that $\mathcal{K}_1, \tau_{l'}, \sigma_1[x_n \mapsto \tau_{l'}] \models \chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$ as well as $\mathcal{K}_0, \pi_l, \sigma_0[x_n \mapsto \pi^l] \models \chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$. The argument that **Duplicator** wins this game follows again with Lemma 5.10 along the same lines as above for the (X)-move.

Suppose lastly that **Spoiler** makes a (U)-move and that he chooses a path π starting at some s_i and $l \in \mathbb{N}$. Again, let $\chi^{m-1,n}_{\pi^j,\sigma_0[x_n \mapsto \pi^j]}$, $0 \leq j \leq l$ be the characteristic formulas of π^j and $\sigma_0[x_n \mapsto \pi^j]$ up to degree $m-1$ and let $\psi := \bigvee_{0 \leq j < l} {\downarrow} x_n.\chi^{m-1,n}_{\pi^j,\sigma_0[x_n \mapsto \pi^j]}$.

We construct the strategy for **Duplicator** such that he chooses a path τ starting at t_i and $l' \in \mathbb{N}$ with $\mathcal{K}_1, \tau^{l'}, \sigma_1[x_n \mapsto \tau^{l'}] \models \chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$ and $\mathcal{K}_1, \tau^k, \sigma_1[x_n \mapsto \tau^k] \models \psi$ for all $0 \leq k < l'$.

We now argue that such a path must exist. First, there must be a state

t reachable from t_i such that $\mathcal{K}_1, t, \sigma_1[x_n \mapsto s] \models \chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$. Otherwise, we would have that $\mathcal{K}_0, s_0, \sigma_0 \models @_{x_i} \mathsf{EF}{\downarrow} x_n.\chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$ but $\mathcal{K}_1, t_0, \sigma_0 \not\models @_{x_i} \mathsf{EF}{\downarrow} x_n.\chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$ which is impossible by assumption.
Secondly, there must be a finite path $\tau_1 \dots \tau_m$ to such a state t such that $\mathcal{K}_1, \tau^k, \sigma_1[x_n \mapsto \tau^k] \models \psi$ for all $0 \leq k < m$. Suppose for a moment this was not the case. This would mean that on every finite path $\tau_1 \dots \tau_m$ such that $\mathcal{K}_1, \tau^m, \sigma_1[x_n \mapsto \tau^m] \models \chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]}$ there is a moment $k < m$ such that $\mathcal{K}_1, \tau^k, \sigma_1[x_n \mapsto \tau^k] \not\models \psi$. Thus, we have that $\mathcal{K}_1, t_0, \sigma_1 \not\models @_{x_i} \mathsf{E}\psi\mathsf{U}({\downarrow} x_n.\chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]})$ and obviously $\mathcal{K}_0, s_0, \sigma_0 \models @_{x_i} \mathsf{E}\psi\mathsf{U}({\downarrow} x_n.\chi^{m-1,n}_{\pi^l,\sigma_0[x_n \mapsto \pi^l]})$ which contradicts the assumption.
The remainder of the (U)-move works in the same way as for the (G)-move, i.e. for every choice of **Spoiler** on **Duplicator**'s path, **Duplicator** picks a point on **Spoiler**'s path that matches the disjunct in ψ that is actually satisfied. In the case that **Spoiler** picks the last point of both paths to continue form, **Duplicator** has no choice anyway. The argumentation why **Duplicator** wins is then also the same as above. This finishes the proof. □

Thus, at least on structures with finite branching-degree, we get that these games exactly characterise the expressive power of HCTL.
However, to prove that there is no formula distinguishing two structures we only need the first direction (Theorem 5.5) which does not pose any restrictions on the structures.

5.2 The Hierarchy on Kripke Structures

We will now study the relative expressive power of all hybrid branching-time logics. This section is loosely divided into the comparisons below the level of HCTL*, i.e. HCTL, HCTL$^+$ and HFCTL$^+$ and the comparisons between all variants of HCTL* and the Hybrid μ-calculus.

5.2.1 Below HCTL$^*_{\mathsf{ss}}$

We first show that – similar to the non-hybrid case – HCTL $\equiv$ HCTL$^+$. Thus, as in the non-hybrid case, adding boolean connectives does not increase the expressive power. The proof is very similar to the proof in the non-hybrid case [29] and was already extended to HCTL$^+$ on tree structures in [52].

Theorem 5.12. For each formula $\varphi \in$ HCTL$^+$ there is a formula $\varphi' \in$ HCTL such that $\varphi \equiv \varphi'$.

Proof. We describe how to transform an HCTL$^+$ formula φ into an equivalent HCTL formula. This can be done by rewriting each path formula in φ. The key observation is that the binder only occurs as a state formula in HCTL$^+$ and thus path formulas are essentially non-hybrid in the sense that they are evaluated relative to a fixed variable interpretation. And fixed variables can simply be regarded as atomic propositions that happen to hold at a single state only. For this reason we proceed as in the non-hybrid case ([29]).
First, using the equivalences $\mathsf{A}\psi \equiv \neg\mathsf{E}\neg\psi$, $\neg\mathsf{X}\psi \equiv \mathsf{X}\neg\psi$, $\neg(\psi_1\mathsf{U}\psi_2) \equiv \mathsf{G}\neg\psi_2 \vee (\neg\psi_2\mathsf{U}(\neg\psi_1 \wedge \neg\psi_2))$ we can rewrite φ such that all path formulas are preceded by an E path-quantifier.
Now path formulas are boolean combinations of state formulas, as well as X-, U- and G-formulas. Using equivalences from propositional logic we can rewrite the path formula into a disjunctive normal form.
Then, using the equivalence $\mathsf{E}(\psi_1 \vee \psi_2) \equiv \mathsf{E}\psi_1 \vee \mathsf{E}\psi_2$, we can split path formulas and assume that remaining path formulas only consist of conjunctions of state formulas as well as X-, U- and G-formulas.
We then use $\mathsf{X}\varphi_1 \wedge \mathsf{X}\varphi_2 \equiv \mathsf{X}(\varphi_1 \wedge \varphi_2)$ and $\mathsf{G}\varphi_1 \wedge \mathsf{G}\varphi_2 \equiv \mathsf{G}(\varphi_1 \wedge \varphi_2)$ to merge X- and G-formulas and the equivalence $\mathsf{E}(\varphi \wedge \psi) \equiv \varphi \wedge \mathsf{E}\psi$ for some state formula φ to remove state formulas directly under a path quantifier.
Thus, we can assume that each path formula has the form

$$\mathsf{E}(\mathsf{X}\Lambda_1 \wedge \bigwedge_{i\in I} \varphi_i \mathsf{U} \chi_i \wedge \mathsf{G}\Lambda_2)$$

with suitable state formulas $\Lambda_1, \Lambda_2, \varphi_i, \chi_i$.
We then obtain an HCTL formula by guessing the order in which the U-formulas are satisfied along such a path. This is done by the following formula:

$$\begin{aligned}
&\Lambda_2 \wedge \bigvee_{J\subseteq I} (\bigwedge_{j\notin J} \chi_j) \wedge (\bigwedge_{j\in J} \varphi_j) \wedge \mathsf{EX}\Big(\Lambda_1 \wedge \\
&\qquad \bigvee_{\pi\in\mathsf{Perm}(J)} \mathsf{E}((\Lambda_2 \wedge \bigwedge_{j\in J} \varphi_{\pi(j)})\mathsf{U}\Big(\chi_{\pi(1)} \wedge \\
&\qquad\qquad \mathsf{E}((\Lambda_2 \wedge \bigwedge_{j\in J, j\neq 1} \varphi_{\pi(j)})\mathsf{U}\Big(\chi_{\pi(2)} \wedge \\
&\qquad\qquad \vdots \\
&\qquad\qquad \mathsf{E}((\Lambda_2 \wedge \varphi_{\pi(|J|)})\mathsf{U}(\chi_{\pi(|J|)} \wedge \mathsf{EG}\Lambda_2))\Big) \dots \Big)
\end{aligned}$$

where $\mathsf{Perm}(J)$ denotes the set of all permutations over J.

Correctness of the construction follows along the same lines of argumentation as in [29]. Essentially (ignoring Next- and Generally-formulas) the argument is as follows: If there is a path satisfying the HCTL$^+$ formula then there is a sequence in which all the Until-formulas are satisfied along this path which is then guessed by the HCTL formula and vice versa. □

In the non-hybrid case, we know that FCTL$^+$ is more expressive than CTL$^+$. For example in [30] it was shown that the formula $\mathsf{EGF}p$ cannot be expressed by CTL$^+$/CTL. A similar result was already shown for HCTL$^+$ interpreted only over computation trees in [52]. This of course also gives us a separation result over general Kripke structures.

Proposition 5.13. There is no formula in HCTL$^+$ that is equivalent to the HFCTL$^+$ formula $\mathsf{EGF}p$.

Quite interestingly this result does not hold anymore if we restrict the logic to be interpreted only over finite structures. On finite structures we can use that an infinite ocurrence of p along some path is equivalent to the occurrence of p at some state that lies on a (finite) loop in the structure. This can easily be expressed already in HCTL and consequently we get that on finite structures the HFCTL$^+$ formula $\mathsf{EGF}p$ is equivalent to the HCTL formula $\mathsf{EF}{\downarrow}x.\mathsf{EF}(p \wedge \mathsf{EXEF}x)$.
By generalising this idea and combining it with the translation from HCTL$^+$ to HCTL in Theorem 5.12, we obtain the following result for HFCTL$^+$ on finite structures.

Theorem 5.14. On finite structures, every HFCTL$^+$ formula is equivalent to an HCTL formula.

Proof. We begin with the same equivalences as in the proof for Theorem 5.12 to obtain a formula in which all path formulas are of the form

$$\mathsf{E}(\mathsf{X}\Lambda_1 \wedge \bigwedge_{i\in I_1} \varphi_i \mathsf{U} \psi_i \wedge \mathsf{G}\Lambda_2 \wedge (\bigwedge_{i\in I_2} \mathsf{GF}\chi_i) \wedge (\bigwedge_{i\in I_3} \neg\mathsf{GF}\xi_i)) \tag{5.1}$$

for suitable HFCTL$^+$ formulas $\varphi_i, \psi_i, \chi_i, \xi_i$. Note that $\neg\mathsf{GF}\xi_i$ essentially means that there is a point from which on ξ_i is not satisfied anymore.
To transform such formulas into HCTL formulas we again guess the order in which all Until-formulas are satisfied – ignoring the fairness constraints for the initial part – and then we guess a point from which on there are cyclic paths along which the χ_i are satisfied but no ξ_i is.
Thus, we get the following translation:

$$\Lambda_2 \wedge \bigvee_{J\subseteq I_1} (\bigwedge_{j\notin J} \psi_j) \wedge (\bigwedge_{j\notin J} \varphi_j) \wedge$$

$$\mathsf{EX}\Big(\Lambda_1 \wedge \bigvee_{\pi\in\mathsf{Perm}(J)} \mathsf{E}((\Lambda_2 \wedge \bigwedge_{j\in J} \varphi_{\pi(j)})\mathsf{U}\Big(\psi_{\pi(1)}\wedge$$

$$\vdots$$

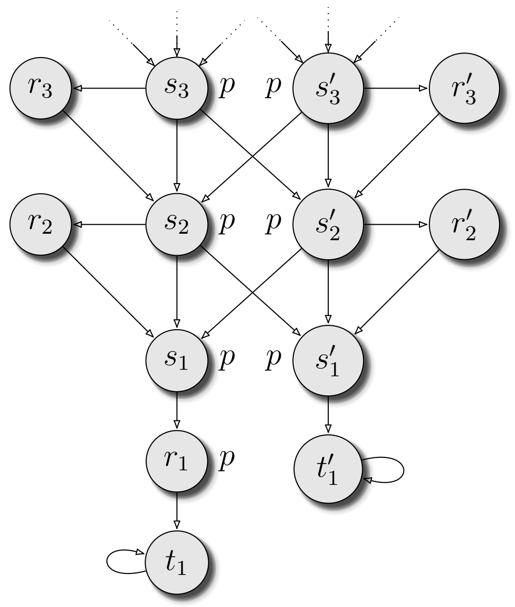
$$\mathsf{E}((\Lambda_2 \wedge \varphi_{\pi(|J|)})\mathsf{U}(\psi_{\pi(|J|)} \wedge \xi)\Big) \ldots\Big)$$

where $\mathsf{Perm}(J)$ denotes the set of all permutations over J and

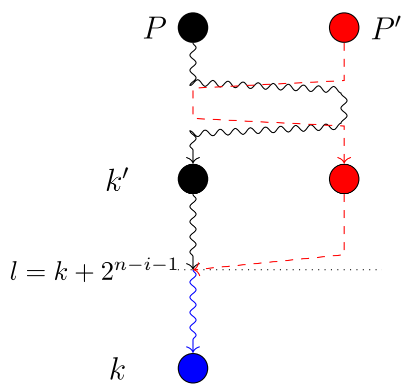
$$\xi := \mathsf{E}(\Lambda_2\mathsf{U}(\Lambda_2 \wedge \downarrow x. \bigwedge_{i\in I_2} \mathsf{E}((\Lambda_2 \wedge (\bigwedge_{i\in I_3} \neg\xi_i))$$
$$\mathsf{U}(\Lambda_2 \wedge (\bigwedge_{i\in I_3} \neg\xi_i) \wedge \chi_i \wedge \mathsf{E}((\Lambda_2 \wedge (\bigwedge_{i\in I_3} \neg\xi_i))\mathsf{U}x))))).$$

Suppose some state satisfies (5.1) in some finite structure. Then there is a path π satisfying all conjuncts of (5.1). We only argue correctness of the translation for the infinite part of the path after all Until-formulas were satisfied and also ignore that Λ_2 is satisfied on every state of the path. Correctness for the initial part follows along the same lines as the translation from HCTL$^+$ to HCTL in Theorem 5.12.
Since the structure is finite there has to be some point x occurring infinitely often along π and also a moment on π such that none of the ξ_i are satisfied from this moment on. Furthermore, since every χ_i is satisfied infinitely often there has to be a part of the path such that χ_i is satisfied on some state between two occurrences of x. Thus, for every i there is a cycle starting at x in the structure along which χ_i is satisfied. Hence, ξ is satisfied. The converse direction follows by a piecewise reconstruction of the whole path with infinitely many occurrences of each cycle satisfying some χ_i. □

We will use this result to show that already on finite structures HCTL$^*_{\mathsf{ss}}$ is more expressive than HFCTL$^+$. We do this in two steps.
First, using the Ehrenfeucht–Fraïssé games for HCTL, we will prove that there are two classes of *finite structures* distinguishable by HCTL$^*_{\mathsf{ss}}$ such that no HCTL formula can distinguish them. Combined with Theorem 5.14 this will also prove that, already on finite structures, HCTL$^*_{\mathsf{ss}}$ is more powerful than HFCTL$^+$. Of course, as for Proposition 5.13 above, this result then also generalises to the class of all Kripke structures.
Interestingly, we can use the same CTL* formula that shows the expressiveness gap between CTL* and FCTL$^+$ but since the hybrid versions of both logics are more expressive we need two more sophisticated classes of structures to show the same result.



(a) Structure $\mathcal{A}$.



(b) Sketch of Duplicator's path choice. The black path is Spoiler's choice. Duplicator's path is depicted in red. The blue part means that both paths have joined.

Figure 5.3: Structure $\mathcal{A}$ and a sketch of Duplicator's path choice on $\mathcal{A}$.

Consider the structure $\mathcal{A}$ in Figure 5.3a. Note that despite $\mathcal{A}$ being an infinite structure as a whole, it is essentially finite from every state because every path traverses the structure downwards and either ends in t_1 or t_1'.
In the following, we refer to the index of a state's name as the level of the structure and the letter of its name as the type of the state. Also note that each path that goes from level i to level $i-1$ either visits s_{i-1} or s_{i-1}'.

Lemma 5.15. For every $i \in \mathbb{N}$ we have that

$$\mathcal{A}, s_i \models \mathsf{AF}(p \wedge \mathsf{X}p) \text{ and } \mathcal{A}, r_i \models \mathsf{AF}(p \wedge \mathsf{X}p),$$

but

$$\mathcal{A}, s_i' \not\models \mathsf{AF}(p \wedge \mathsf{X}p) \text{ and } \mathcal{A}, r_i' \not\models \mathsf{AF}(p \wedge \mathsf{X}p).$$

Proof. To see this, note that every path starting on the left-hand side of the structure either ends in the left bottom component, where at s_1 and r_1 we have that two p's hold, or the path changes side at some point going from some s_j to s_{j-1}' and seeing two p's there.

However, the zig-zag path starting somewhere on the right-hand side of $\mathcal{A}$ that always goes from s'_j to r'_j and then to s'_{j-1} etc. until it loops at t'_1, never sees two consecutive p's. $\square$

This lemma gives us a natural partition of $\mathcal{A}$ into "substructures" that start on the left-hand side and "substructures" that start on the right-hand side. We will now prove that for sufficiently large structures there is no HCTL formula that can distinguish these structures.

Theorem 5.16. Let $n \in \mathbb{N}$ and $m = 2^{n+1}$. Duplicator wins $\mathcal{G}^n_{\mathrm{HCTL}}(\mathcal{A}, s_m, \mathcal{A}, s'_m)$.

Proof. To prove this, we describe a winning strategy for Duplicator. Suppose that i rounds have been played already. To win, Duplicator maintains the following invariant throughout the game:

- The pebbles placed by Duplicator are always on the same level and of the same type as Spoiler's pebble in the same round.
- There is some $k \geq 2^{n-i}$ such that each pair of pebbles placed by Spoiler and Duplicator in the same round on level k or smaller marks exactly the same state. Furthermore, pairs of pebbles placed above level k are on opposing sides in the structure.
- The first pair of pebbles placed above level k is at least on level $k+2^{n-i}$.

It is obvious that if Duplicator can maintain this invariant for n rounds then she wins.
Furthermore, at the beginning of the game the invariant holds with $k = 2^n$. Suppose now that i rounds have already been played according to this strategy for some $i \leq n$ and Spoiler decides to move from some pebble p that has previously been placed. Let p' be the corresponding pebble placed in the same round as p. Since up to now Duplicator has maintained the invariant above, p' is on the same level and of the same type as p. Spoiler now has three types of moves available.
Suppose first that he chooses to make an (X)-move. There are two scenarios. First, p and p' may lie above k, thus by the second point of the invariant one is on the left-hand side of the structure and one is on the right-hand side. Duplicator then simply picks the unique successor of p' that matches Spoiler's choice in level and type but is on the opposing side of the structure. Secondly, if p, p' lie at k or below, then they mark the exact same state and Duplicator simply mimics Spoiler's move exactly. In both cases the invariant is maintained, even with the same k.

Suppose now that Spoiler chooses to make a (G)-move. Again, if p, p' are below level k, Duplicator can simply mimic Spoiler's path and also Spoiler's pick because p and p' mark the same state. So, suppose further that p, P' are above level k. Thus, p and p' are on opposite sides of the structure and by the third point of the invariant then p, p' are on a level greater than or equal to $k + 2^{n-i}$.
Let $k' \geq k + 2^{n-i}$ be the lowest level above k such that pebbles are placed on level k' and let $l := k + 2^{n-i-1}$. After Spoiler chooses his path, Duplicator chooses her path as follows:

- Up to level $l + 1$ she simply mimics Spoiler's path but stays on the opposite side of the structure.
- At state s_{l+1} or s'_{l+1} – which Spoiler's path necessarily visits and no matter if Spoiler's path goes through r_{l+1} resp. r'_{l+1} – Duplicator's path changes sides to meet up with Spoiler's path at s_l or s'_l depending on which one of these Spoiler's path visits.
- From then on Duplicator's path simply follows exactly Spoiler's path.

A rough illustration of Duplicator's path is depicted in Figure 5.3b. Spoiler can now pick some pebble on Duplicator's path. Note that Duplicator's choice after picking her path is entirely dependent on Spoiler's choice by the first part of the invariant. Thus we only need to show that the invariant is preserved, no matter what Spoiler chooses to play. We distinguish two cases:

- First, assume that Spoiler picks some state above l. In this case k stays the same for the next round and the first pair of pebbles is placed at least $2^{n-i-1} + 1$ levels above k.
- Secondly, assume that Spoiler picks a state on level l or below. In this case we pick the new k to be the level on which both new pebbles are placed. The first pair of pebble above the newly placed ones is on level k' and thus at least $k' - l \geq (k + 2^{n-i}) - (k + 2^{n-i-1}) = 2^{n-i-1}$ levels above the new pebbles.

Suppose lastly that Spoiler makes a (U)-move. Here, Spoiler can also choose an endpoint on the path and after Duplicator has chosen her path and endpoint, Spoiler can potentially force the game to continue from there. There are two cases.
First, suppose that Spoiler picks his path and endpoint in such a way that his endpoint is not at the state r_{l+1} or r'_{l+1}. Then Duplicator chooses her path in the same way as described for (G)-moves and her endpoint simply on the

same level as Spoiler's endpoint. No matter which option Spoiler chooses, the invariant is maintained with the same arguments as for (G)-moves.
So, suppose that Spoiler picks a path from some pebble p with endpoint at r_{l+1} or r'_{l+1}. Thus, p lies above level k and there is some pebble p' on the opposing side of the structure that is on the same level and marks the same type of state. In this case Duplicator cannot simply ignore r_{l+1}/r'_{l+1} and change sides to meet Spoiler's path since Spoiler could then force the game exactly to this point. However, by picking r_{l+1} or r'_{l+1} as the endpoint Spoiler has limited himself for the second part of the (U) move and can only force the game to stay somewhere above l. Thus Duplicator simply chooses a path that remains on the opposing side of Spoiler's path up to level l.
Suppose Spoiler chooses to play the endpoints at r_{l+1} and r'_{l+1}. Then with the same k as in the previous round we get that the first pebbles above level k are at least 2^{n-i-1} levels above k. And lastly, if Spoiler chooses to play some level above $l+1$ the invariant is also maintained with the same argument.
In any case, Duplicator can maintain the invariant and thus she wins the game $\mathcal{G}^n_{\text{HCTL}}(\mathcal{A}, s_{2^{n+1}}, \mathcal{A}, s'_{2^{n+1}})$. □

Theorem 5.17. There is no HFCTL$^+$ formula that is logically equivalent to the CTL* formula $\mathsf{AF}(p \wedge \mathsf{X}p)$.

Proof. Suppose there was such a formula. By Theorem 5.14 this formula is equivalent to an HCTL formula φ on finite structures. Let $\mathsf{nd}(\varphi) = n$. Then by Theorems 5.5 and 5.16 the formula φ cannot distinguish between the states $s_{2^{n+1}}$ and $s'_{2^{n+1}}$. However, by Lemma 5.15 we have that $\mathcal{A}, s_{2^{n+1}} \models \mathsf{AF}(p \wedge \mathsf{X}p)$ but $\mathcal{A}, s'_{2^{n+1}} \not\models \mathsf{AF}(p \wedge \mathsf{X}p)$. □

Thus, we get that CTL* and HFCTL$^+$ are incomparable and since HCTL$^*_{\mathsf{ss}}$ is an extension of CTL* and HFCTL$^+$ we obtain the following:

Corollary 5.18. Already on finite structures HCTL$^*_{\mathsf{ss}}$ is more expressive than HFCTL$^+$.

Thus, we have established a hierarchy below HCTL$^*_{\mathsf{ss}}$, namely

$$\text{HCTL} \equiv \text{HCTL}^+ \prec \text{HFCTL}^+ \prec \text{HCTL}^*_{\mathsf{ss}}$$

that looks quite similar to the non-hybrid version:

$$\text{CTL} \equiv \text{CTL}^+ \prec \text{FCTL}^+ \prec \text{CTL}^*.$$

Quite interestingly, the last result which states that HCTL$^*_{\mathsf{ss}}$ is more expressive than HFCTL$^+$ does not hold anymore if we restrict our attention only to trees.

Example 5.19. On tree structures we have that the HFCTL$^+$ formula $\downarrow s.\mathsf{AF}(p \wedge \downarrow x.\, @_s\, \mathsf{EF}(\mathsf{EX}x \wedge p))$ (in fact it is even an HCTL formula) is equivalent to the CTL* formula $\mathsf{AF}(p \wedge \mathsf{X}p)$.
The HFCTL$^+$ formula exploits the property of trees that each state has a unique parent node. Thus instead of looking for a state on each path such that the state itself and its successor both satisfy p the HFCTL$^+$ formula looks for a state satisfying p such that its unique parent node also satisfies p. The identification of the parent node can easily be done using hybrid operators.

It is open whether HCTL$^*_{\mathsf{ss}}$ interpreted only on trees is more expressive than HFCTL$^+$.

5.2.2 HCTL* and the Hybrid μ-calculus

In this section we try to establish a connection between the three variants of HCTL* and H$_\mu$. While we know that CTL* can be translated into the modal μ-calculus this is a priori not clear for their hybrid extensions. While both extensions feature the same additional hybrid operators, their semantical interpretation differs quite a bit.
The first step in establishing such a connection is to translate HCTL$^*_{\mathsf{ss}}$ formulas into H$_\mu$ and by doing so we show that H$_\mu$ is at least as expressive as HCTL$^*_{\mathsf{ss}}$.
However, already the translation from CTL* to L$_\mu$ is non-trivial. It is helpful to recall the basic ideas of this translation: the key-difference between both logics is found in the path quantifiers of CTL*. In branching-time logics it is possible to quantify over an infinite path and then state conditions that need to hold on this infinite path. This is not directly possible in L$_\mu$. Instead, there we have to generate such an infinite behaviour step by step via fixed points.
Thus, it is not surprising that the main challenge is to translate arbitrarily nested path formulas. The key-idea for this part is to see that path formulas in the non-hybrid case can be regarded as LTL formulas with nested CTL* state formulas. Ignoring these nested state formulas for the moment, we can translate the LTL parts of the formula into a Büchi automaton on ω-words that accepts a path if and only if this path satisfies the LTL formula. This automaton in turn can then be translated into a μ-calculus formula that simulates the automaton. The embedded state formulas in the LTL formula can be handled by a decomposition method as usual for CTL*.

HCTL$^*_{\mathsf{ss}}$ and H$_\mu$. We will use a similar approach to translate HCTL$^*_{\mathsf{ss}}$ to H$_\mu$. Inspecting the grammar for HCTL$^*_{\mathsf{ss}}$, we see that the path formulas are syntactically also *only* LTL formulas with embedded state formulas. Hybrid operators in this variant are restricted to state formulas only, which means that a path formula is evaluated with respect to a fixed variable interpretation. However, these path formulas feature an extended vocabulary. For example the path formula $\mathsf{F}x$ in the HCTL$^*_{\mathsf{ss}}$ formula $\downarrow x.\mathsf{EF}x$ needs to deal with free variables that may occur on a path formula. And thus, a Büchi automaton that checks for the occurrence of x along some path in the structure also needs to take care of these variables in some form.
To get started we first need a few definitions.

Definition 5.20. A path formula ψ is called *pure* if there are no occurrences of path quantifiers E, A or hybrid operators $\downarrow, @$ in ψ.

Pure path formulas in particular are evaluated with respect to a fixed variable assignment. Thus, we can simply treat free variables like atomic propositions. To make this intuition precise we code them into the labeling of a structure.

Definition 5.21. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure and $\sigma : \mathit{Var} \rightarrow S$ be a variable assignment. We define $\mathcal{K}_\sigma := \langle S, \rightarrow, L'\rangle$ with $L' : \mathit{Prop} \cup \mathit{Var} \rightarrow 2^S$ with $L'(p) = L(p)$ for all $p \in \mathit{Prop}$ and $p \in \mathit{Nom}$ and $L'(x) = \{\sigma(x)\}$ for all $x \in \mathit{Var}$.

Thus, $\mathcal{K}_\sigma$ extends a Kripke structure $\mathcal{K}$ with fresh atomic propositions for each variable and they hold at exactly the states assigned by σ. Since $\mathcal{K}$ and $\mathcal{K}_\sigma$ only differ in their labeling, paths in $\mathcal{K}$ can also be regarded as paths in $\mathcal{K}_\sigma$ and vice versa.
However, not all paths over the extended alphabet $\mathit{Prop} \cup \mathit{Var}$ do encode a proper path from such an extended Kripke structure. For example a path $\pi = s_0 s_1 s_2 \ldots$ over $\mathit{Prop} \cup \mathit{Var}$ with $s_0 \neq s_1$ and $L(x) = \{s_0, s_1\}$ obviously cannot be traced back to an extended Kripke structure since x does not encode a "hybrid" state variable that only marks a single state.

Definition 5.22. Let π be a path over $\mathit{Prop} \cup \mathit{Var}$ with states from a set S and $\sigma : \mathit{Var} \rightarrow S$ a variable assignment. We call π *consistent with respect to σ* if for all $i \in \mathbb{N}$ and all $x \in \mathit{Var}$ it holds that $\pi^i \in L(x)$ if and only if $\pi^i = \sigma(x)$.

The following lemma about consistent paths and extended Kripke structures is easy to see.

Lemma 5.23. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure, $\sigma : \mathit{Var} \rightarrow S$ a variable assignment and π be a path in $\mathcal{K}_\sigma$. Then π is consistent with respect to σ.

We will now prove a connection between pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formulas and LTL formulas. Since pure path formulas and LTL formulas are syntactically the same – aside from the fact that variables may be treated as atomic propositions – we simply use the index LTL for the satisfaction relation $\models$, i.e. $\models_{\mathrm{LTL}}$, to indicate that a pure path formula is interpreted as an LTL formula.

Lemma 5.24. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure, π a path in $\mathcal{K}$, $\sigma : \mathit{Var} \rightarrow S$ a variable assignment and $k \in \mathbb{N}$.
Then for every pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formula ψ over atomic propositions *Prop* and variables from *Var* it holds that $\mathcal{K}, \pi, k, \sigma \models \psi$ if and only if $\mathcal{K}_\sigma, \pi^{[k,\infty)} \models_{\mathrm{LTL}} \psi$.

Proof. We will prove this by a straightforward induction on ψ. For this, suppose that $\mathcal{K}, \pi, k, \sigma \models \psi$.
First, assume that $\psi = \varphi$ for some state formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$. Since ψ is pure this means that φ is a boolean combination of propositions and variables. And because π – regarded as a path in $\mathcal{K}_\sigma$ – is consistent with σ according to Lemma 5.23 and all states in $\mathcal{K}_\sigma$ and $\mathcal{K}$ agree on their atomic propositions, we also get that $\mathcal{K}_\sigma, \pi^{[k,\infty)} \models_{\mathrm{LTL}} \psi$.
The claim for negation and disjunction follows immediately. So, suppose $\psi = \mathsf{X}\psi'$. Then $\mathcal{K}, \pi, k+1, \sigma \models \psi'$. By the induction hypothesis we then get that $\mathcal{K}_\sigma, \pi^{[k+1,\infty)} \models_{\mathrm{LTL}} \psi'$. By the semantics of LTL formulas we also obtain that $\mathcal{K}_\sigma, \pi^{[k,\infty)} \models_{\mathrm{LTL}} \psi$.
Lastly, suppose that $\psi = \psi_1 \mathsf{U} \psi_2$. Then there is some $j \in \mathbb{N}$ such that $\mathcal{K}, \pi, k+j, \sigma \models \psi_2$ and for all $i = 0, \ldots, j-1$ we have that $\mathcal{K}, \pi, k+i, \sigma \models \psi_1$. By the induction hypothesis we get that $\mathcal{K}_\sigma, \pi^{[k+j,\infty)} \models_{\mathrm{LTL}} \psi_2$ and $\mathcal{K}_\sigma, \pi^{[k+i,\infty)} \models_{\mathrm{LTL}} \psi_1$ for all $0 \leq i < j$ and thus also $\mathcal{K}_\sigma, \pi^{[k,\infty)} \models_{\mathrm{LTL}} \psi$ by the semantics of LTL. □

Because we have no past-operators and the jump is not allowed in pure path formulas there is no way to look back into the past in a pure path formula and hence when interpreting a pure path formula as an LTL formula we can restrict the attention to the suffix of a path starting at the current moment. Furthermore, we can treat pure path formulas – which are evaluated with respect to a fixed variable interpretation – just like LTL formulas over an extended structure.
Next we extend the well-known connection between LTL formulas and Büchi-automata to encompass pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formulas.

Theorem 5.25. For each pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formula ψ of size n over atomic propositions *Prop* and variables $\{x_1, \ldots, x_k\}$ there is a Büchi automaton $\mathcal{A}_\psi$ of size $\mathcal{O}(n \cdot 2^n)$ such that:

For all Kripke structures $\mathcal{K} = \langle S, \rightarrow, L\rangle$ over $\mathit{Prop} \cup \{x_1, \ldots, x_k\}$ and all paths π over $\mathcal{K}$, the path π is accepted by $\mathcal{A}_\psi$ if and only if $\pi \models_{\mathrm{LTL}} \psi$.

Proof. To construct $\mathcal{A}_\psi$ we first observe that ψ is an LTL formula with possibly added variable tests. We treat variable tests for the moment like atomic propositions and construct a Büchi automaton that accepts a path π if and only if π satisfies ψ (as an LTL formula) [93]. The construction then immediately yields the size estimation of the automaton as well as the rest of the statement. $\square$

Note that the constructed Büchi-automaton $\mathcal{A}_\psi$ as well as the (LTL)-formula ψ in general accept "more" paths than the pure hybrid path formula ψ since they only check the sequence of propositions and treat variables like any other proposition. For example, $\mathcal{A}_\psi$ also accepts paths that are not consistent with respect to any variable assignment, i.e. paths in which a "proposition" x may occur at more than one state.
However, on Kripke structures that are extended in a "hybrid" way the automaton works as intended. We will only use the automaton on such structures and also only use it as an intermediate step in the translation of $\mathrm{HCTL}^*_{\mathsf{ss}}$ into the fully hybrid μ-calculus. The next goal is to translate such a Büchi-automaton into an H_μ-formula.
It is well-known that each Büchi-automaton can be simulated by an L_μ formula [24]. However, to use this formula we also need to bridge the small gap between L_μ over an extended "hybrid" vocabulary and H_μ similar to how LTL and pure path formulas are connected in Lemma 5.24.
Similar to the LTL case we write $\mathcal{K}, s \models_{\mathrm{L}_\mu} \varphi$ to indicate that φ is interpreted as a purely modal μ-calculus formula with free first-order variables regarded as propositions and assuming that no binders and jumps occur in φ.

Lemma 5.26. For each H_μ-formula φ without any occurrence of $\downarrow x.\psi$ or $@_x\,\psi$ in it, it holds that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}_\sigma, s \models_{\mathrm{L}_\mu} \varphi$.

Proof. (Sketch) To prove this by induction on φ we need to strengthen the hypothesis in order to deal with free second-order variables. Let $\varphi(X_1, \ldots, X_m)$ be a formula with free second-order variables $X_1, \ldots, X_m$ and $\rho : \{X_1, \ldots, X_m\} \rightarrow 2^{S\times(\mathit{Var}\rightarrow S)}$ be an interpretation for them. We define $\rho' : \{X_1, \ldots, X_m\} \rightarrow 2^S$ to be $\rho'(X) := \{s \mid (s, \sigma) \in \rho(x)\}$. We now show by induction on φ that $\mathcal{K}, s, \sigma, \rho \models \varphi(X_1, \ldots, X_m)$ if and only if $\mathcal{K}_\sigma, s, \rho' \models_{\mathrm{L}_\mu} \varphi(X_1, \ldots, X_m)$.
The case for $\varphi = p$ holds because $\mathcal{K}$ and $\mathcal{K}_\sigma$ agree everywhere on all atomic propositions. The case for $\varphi = x$ holds by construction of $\mathcal{K}_\sigma$ and the case for $\varphi = X$ follows by construction of ρ'. Boolean combinations as well as modal operators follow by simple semantical arguments. And finally, for the case of

$\varphi = \mu X.\psi(X)$ we use the characterisation of least fixed points as the union of its approximations. It is straightforward to show by a separate induction that the statement holds for all approximations. $\square$

Theorem 5.27. For each Büchi automaton $\mathcal{A}$ over $Prop \cup \{x_1, \ldots, x_k\}$ of size m there is an H_μ formula $\varphi_{\mathcal{A}}$ of size at most $O(m \cdot 2^m)$ such that $\mathcal{K}, s, \sigma \models \varphi_{\mathcal{A}}$ if and only if there is a path π in $\mathcal{K}_\sigma$ starting at s such that $\mathcal{A}$ accepts π.

Proof. It is well-known that for each Büchi automaton $\mathcal{A}$ of size m there is an L_μ-formula $\varphi'_{\mathcal{A}}$ in vectorial form such that $\mathcal{K}_\sigma, s \models_{\mathrm{L}_\mu} \varphi'_{\mathcal{A}}$ if and only if there exists a path π starting at s such that $\mathcal{A}$ accepts π, c.f. [8]. This formula $\varphi'_{\mathcal{A}}$ is of size $O(m)$. We then transform $\varphi'_{\mathcal{A}}$ into an equivalent (non-vectorial) L_μ-formula $\varphi_{\mathcal{A}}$ which involves a blowup of size $O(m \cdot 2^m)$, c.f. [17].
With Lemma 5.26 and the fact that $\varphi_{\mathcal{A}}$ does not have any occurrences of $\downarrow x$ or $@_x$ because it is an L_μ formula (extended with possible variable tests that are treated like ordinary propositions) we get that $\mathcal{K}_\sigma, s \models_{\mathrm{L}_\mu} \varphi_{\mathcal{A}}$ if and only if $\mathcal{K}, s, \sigma \models \varphi_{\mathcal{A}}$. $\square$

We are now ready to show the main result of this section.

Theorem 5.28. For each formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ there is a formula $\varphi' \in \mathrm{H}_\mu$ such that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}, s, \sigma \models \varphi'$ for all Kripke structures $\mathcal{K} = \langle S, \to, L \rangle$, states $s \in S$ and variable assignments $\sigma : Var \to S$.

Proof. First, we will give a translation for $\mathrm{HCTL}^*_{\mathsf{ss}}$ formulas and then argue for correctness of the translation.
Suppose $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ over the variables $\{x_1, \ldots, x_k\}$. The cases up to path formulas are straightforward:

$$\begin{aligned} \tau(p) &:= p & \tau(\varphi \vee \chi) &:= \tau(\varphi) \vee \tau(\chi) \\ \tau(x) &:= x & \tau(\downarrow x.\varphi) &:= \downarrow x.\tau(\varphi) \\ \tau(\neg\varphi) &:= \neg\tau(\varphi) & \tau(@_x\, \varphi) &:= @_x\, \tau(\varphi) \end{aligned}$$

For the case of $\varphi = \mathsf{E}\psi$, let $\{\varphi_1, \ldots, \varphi_m\}$ be the maximal state-subformulas in ψ, i.e. subformulas that start with $\mathsf{E}, \mathsf{A}, \downarrow x.$ or $@_x$, and without loss of generality suppose that φ features the variables $\{x_1, \ldots, x_k\}$.
We first replace those by fresh atomic propositions p_{φ_i}. The resulting formula is a pure $\mathrm{HCTL}^*_{\mathsf{ss}}$ path formula over the propositions $Prop \cup \{p_{\varphi_1}, \ldots, p_{\varphi_m}\}$ and variables $\{x_1, \ldots, x_k\}$. Then, according to Theorem 5.25, we construct a Büchi-automaton $\mathcal{A}_\psi$ for this formula. Furthermore, according to Theorem 5.27 there is a H_μ formula $\varphi_{\mathcal{A}_\psi}$ that simulates this Büchi-automaton and thus the formula ψ.

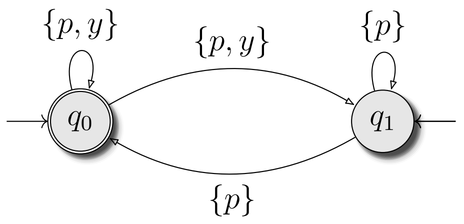
Finally, we translate the remaining maximal state subformulas $\{\varphi_1, \ldots, \varphi_m\}$ recursively. Let $\tau(\varphi_1), \ldots, \tau(\varphi_m)$ be their respective translations. We obtain the final translated formula by replacing the atomic propositions p_{φ_i} in $\varphi_{\mathcal{A}_\psi}$ by their respective translations. Thus:

$$\tau(\mathsf{E}\psi) := \varphi_{\mathcal{A}_\psi}\left[\tau(\varphi_1)/p_{\varphi_1}, \ldots, \tau(\varphi_m)/p_{\varphi_m}\right].$$

It remains to be shown that this translation is correct. For this, let $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure, $s \in S$ and $\sigma : \mathit{Var} \to S$ a variable assignment. We prove that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}, s, \sigma \models \tau(\varphi)$ by induction on φ.
The only interesting case is $\varphi = \mathsf{E}\psi$. Suppose first that ψ is a pure path formula and that $\mathcal{K}, s, \sigma \models \varphi$. Then there is a path π in $\mathcal{K}$, starting at s such that $\mathcal{K}, \pi, 0, \sigma \models \psi$. By Lemma 5.24 we get that $\mathcal{K}_\sigma, \pi \models_{\text{LTL}} \psi$. By Theorem 5.25, π is accepted by $\mathcal{A}_\psi$. And finally, by Theorem 5.27, we get that $\mathcal{K}, s, \sigma \models \varphi_{\mathcal{A}_\psi}$.
For the case that ψ is not pure, we additionally need the fact that $\mathcal{K}, s, \sigma \models \varphi[\chi/p] \Leftrightarrow \mathcal{K}', s, \sigma \models \varphi$ where $\mathcal{K}'$ extends $\mathcal{K}$ with an atomic proposition p such that $p \in L(s) \Leftrightarrow \mathcal{K}, s, \sigma \models \chi$. This can be shown by a straightforward induction on φ, both for $\text{HCTL}^*_{\mathsf{ss}}$ and H_μ. $\square$

To illustrate the translation we give a short example.

Example 5.29. Consider the formula $\chi := \downarrow y.\mathsf{EG}\,(\mathsf{F}y \wedge \downarrow x.\mathsf{EXF}x)$. The formula states that there is a path whose starting point is seen infinitely often along the path and at every point of the path there is another path that loops back to the current point.
We first begin by extracting the maximal state-subformula $\downarrow x.\mathsf{EXF}x$ which yields the formula $\downarrow y.\mathsf{EG}\,(\mathsf{F}y \wedge p)$.
We then construct a Büchi automaton $\mathcal{A}$ for the LTL-formula $\mathsf{G}\,(\mathsf{F}y \wedge p)$ over the atomic propositions p and y. The automaton has two states only and simply checks that y is seen infinitely often and p is seen at every moment along the path:



This Büchi automaton can then be translated into the following μ-calculus formula:

$$\varphi := \big[\nu Y.(p \wedge y \wedge \Diamond Y) \vee (p \wedge y \wedge \Diamond \mu Z.(p \wedge \Diamond Z) \vee (p \wedge \Diamond Y))\big] \vee$$

$$\mu Z.(p \wedge \Diamond Z) \vee$$
$$(p \wedge \Diamond \nu Y.(p \wedge y \wedge \Diamond Y) \vee (p \wedge y \wedge \Diamond \mu Z.(p \wedge \Diamond Z) \vee (p \wedge \Diamond Y)).$$

We obtain two fixed point variables, one for each state. The first line describes an accepting run starting at q_0 and the second and third line describe a run starting at q_1. We obtain the translation $\downarrow y.\mathsf{EG}\,(\mathsf{F}y \wedge p)$ as $\downarrow y.\varphi$.
The same procedure can be applied to the replaced subformula $\downarrow x.\mathsf{EXF}x$: Since there are no maximal state-subformulas left we can directly construct an automaton for the LTL formula $\mathsf{XF}x$ and then translate it into a μ-calculus formula. Putting these together yields the translation of $\downarrow x.\mathsf{EXF}x$ as $\downarrow x.\Diamond \mu X.x \vee \Diamond X$.
To finish the translation we put both translations together and obtain that $\tau(\chi) = \varphi[(\downarrow x.\Diamond \mu X.x \vee \Diamond X)/p]$.

To summarise this section we have obtained the following relationship between $\mathrm{HCTL}^*_{\mathsf{ss}}$ and H_μ.

Corollary 5.30. $\mathrm{HCTL}^*_{\mathsf{ss}} \preceq \mathrm{H}_\mu$.

A closer look at the translation also reveals that the translation from $\mathrm{HCTL}^*_{\mathsf{ss}}$ does not introduce new variables. In fact, the hybrid operators binder and jump get translated without any change. Thus, the translation of a formula $\varphi \in \mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ss}}$ falls into H^k_μ. From this and Theorem 4.13, which states that H^k_μ is invariant under k-bisimulation, we get the following:

Corollary 5.31. $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{ss}}$ is invariant under k-bisimulation.

The inclusion in terms of expressive power between $\mathrm{HCTL}^*_{\mathsf{ss}}$ and H_μ immediately raises the question whether it is strict. Similar results for CTL^* and L_μ suggest that this is probably the case. And indeed it is. We will now proceed to prove an even stronger result: there is a formula in L_μ that cannot even be expressed in $\mathrm{HCTL}^*_{\mathsf{pp}}$ and thus neither in $\mathrm{HCTL}^*_{\mathsf{ss}}$.
The idea of this strictness proof is quite simple. We will show that the L_μ formula $\mu X.p \vee \Diamond\Diamond X$ which characterises reachability in an even number of steps is not expressible in $\mathrm{HCTL}^*_{\mathsf{pp}}$. It is well-known that this formula is not expressible in CTL^* (c.f. [24]) because CTL^* essentially lacks an unrestricted recursion mechanism which is intuitively still the case in $\mathrm{HCTL}^*_{\mathsf{pp}}$.
As before we prove the non-expressibility on a restricted class of structures. The result then transfers to all structures. In this case we choose *linear structures or word structures.*

Lemma 5.32. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a word structure and $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$. Let φ' be the LTL formula that is obtained by syntactically removing all path quantifiers A, E from φ. Then it holds that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}, s, \sigma \models \varphi'$.

This is a simple observation because on word structures there is exactly one path through the structure and hence there is no difference between existential and universal path quanifiers.
Thus, Lemma 5.32 essentially states that, on word structures, $\mathrm{HCTL}^*_{\mathsf{pp}}$ is as expressive as hybrid LTL.

Theorem 5.33. There is no $\mathrm{HCTL}^*_{\mathsf{pp}}$ formula that can express the H_μ property $\mu X.\, p \lor \Diamond\Diamond X$.

Proof. Suppose for the sake of contradiction that such a formula φ exists. Then by Lemma 5.32 we get that there is a hybrid LTL formula that characterises reachability in an even number of steps on word structures. However, hybrid LTL can be translated into first-order logic on word structures [38] which, in turn, cannot express this property, c.f. [25]. Thus, such a formula cannot exist. □

Combined with Corollary 5.30 we obtain that H_μ is strictly more expressive than $\mathrm{HCTL}^*_{\mathsf{ss}}$.

Corollary 5.34. $\mathrm{HCTL}^*_{\mathsf{ss}} \prec \mathrm{H}_\mu$.

$\mathrm{HCTL}^*_{\mathsf{ps}}$ and H_μ. Following this, the next question that naturally arises is: is $\mathrm{HCTL}^*_{\mathsf{ps}}$ also subsumed by H_μ? The simple trick of interpreting a path formula with regard to a fixed variable assignment as an LTL formula does not work anymore since now with the binder we can change variable assignments along some path.
In fact, we will now show that this enables us to express properties that are not even expressible in H_μ.

Theorem 5.35. There is no formula $\varphi \in \mathrm{H}_\mu$ such that φ is logically equivalent to the $\mathrm{HCTL}^*_{\mathsf{ps}}$ formula $\mathsf{EG}{\downarrow}x.\mathsf{XG}\neg x$.

Proof. The formula $\varphi_\infty := \mathsf{EG}{\downarrow}x.\mathsf{XG}\neg x$ expresses that there is a path on which one can mark any state and then never see this exact state again. Thus, it is only satisfied on structures that contain an infinite non-looping path.
Let $C_k = \langle S_k, \to, L\rangle$ be the complete clique over k states with no propositions and no nominals and $C_\infty = \langle S_\infty, \to, L\rangle$ be the complete clique over $\mathbb{N}$. It is obvious that we have $C_k, s \not\models \varphi_\infty$ for any $k \in \mathbb{N}$ and any state $s \in S_k$ and $C_\infty, t \models \varphi_\infty$ for any state $t \in S_\infty$.
By Corollary 4.14 we already know that there is no formula in H_μ that can distinguish all finite cliques from the infinite clique. This is because of the invariance under k-bisimulations of the bounded fragments. Thus, there cannot be any formula in H_μ that is equivalent to φ_∞. □

This immediately gives us two follow-up results. The first is that $\mathrm{HCTL}^*_{\mathsf{ps}}$ is more expressive than $\mathrm{HCTL}^*_{\mathsf{ss}}$. This follows by combining Corollary 5.34 and Theorem 5.35.

Corollary 5.36. $\mathrm{HCTL}^*_{\mathsf{ss}} \prec \mathrm{HCTL}^*_{\mathsf{ps}}$.

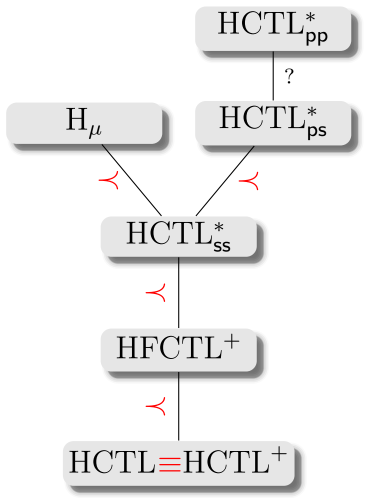
The second observation follows directly from Theorem 5.33 combined with Theorem 5.35.

Corollary 5.37. H_μ and $\mathrm{HCTL}^*_{\mathsf{ps}}$ are incomparable in terms of expressive power.

$\mathbf{HCTL}^*_{\mathsf{pp}}$. We finish this section with only a remark about $\mathrm{HCTL}^*_{\mathsf{pp}}$. The precise relationship between $\mathrm{HCTL}^*_{\mathsf{pp}}$ and $\mathrm{HCTL}^*_{\mathsf{ps}}$ is still open. On the one hand jumps on path formulas seem not that powerful since they "only" jump back to already visited states. But on the other hand we cannot simply eliminate them either. Before jumping back to an already visited state it is possible to place new variables. Thus, after jumping back the information we possess about the path we are looking at has changed compared to before.
One possibility to obtain a clearer picture of their relative expressive power would be to strengthen the EF games introduced in Section 5.1 for HCTL with the aim to obtain a game-theoretic characterisation of the expressive power of $\mathrm{HCTL}^*_{\mathsf{ps}}$ and potentially also $\mathrm{HCTL}^*_{\mathsf{pp}}$. With such a tool it would possibly be easier to grasp the differences in expressive power between both logics – if there are any.

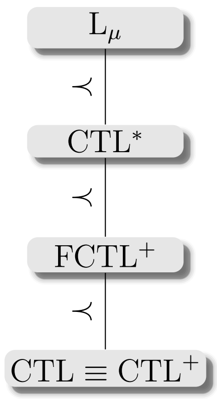
5.2.3 The Hierarchy so far

The results on the relative expressive power of all hybrid branching-time logics are summarised in Figure 5.4a. We see that, especially below $\mathrm{HCTL}^*_{\mathsf{ss}}$, the picture is quite similar to the one for the non-hybrid hierarchy depicted in Figure 5.4b: the hybrid variant of CTL^+ is as expressive as the hybrid variant of CTL and adding fairness constraints and arbitrary nesting of path formulas then adds considerable expressive power.
Only above $\mathrm{HCTL}^*_{\mathsf{ss}}$ and above we see some differences compared to the non-hybrid hierarchy. Quite surprisingly, we have proven that contrary to the non-hybrid logics, the fully hybrid μ-calculus is not the ultimate hybrid logic that subsumes all other logics. Already $\mathrm{HCTL}^*_{\mathsf{ps}}$ can express properties that are not expressible in H_μ. The interaction between a predefined infinite path and a changing variable assignment on this path cannot be simulated by any combination of variables and fixed points in H_μ.



$\mathrm{HCTL}^*_{\mathsf{pp}}$
?
H_μ
$\mathrm{HCTL}^*_{\mathsf{ps}}$
$\prec$
$\prec$
$\mathrm{HCTL}^*_{\mathsf{ss}}$
$\prec$
HFCTL^+
$\prec$
$\mathrm{HCTL} \equiv \mathrm{HCTL}^+$

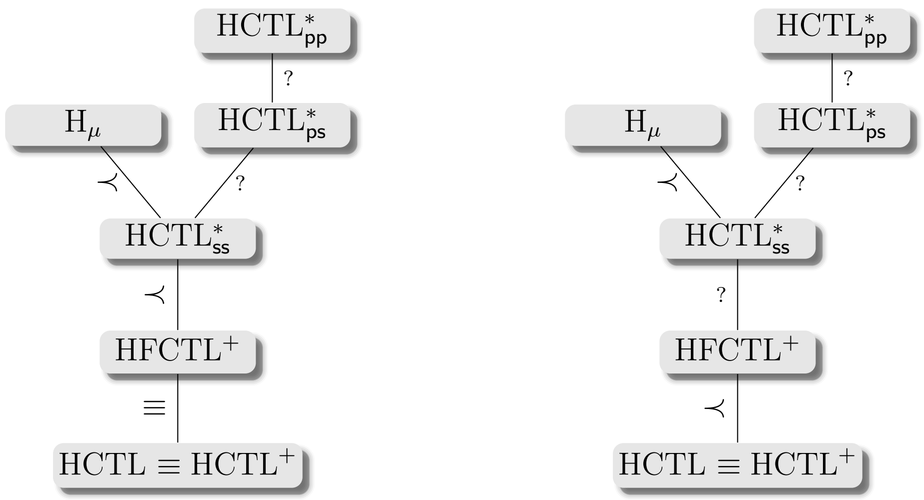
(a) The hybrid branching-time hierarchy.



(b) The branching-time hierarchy.

Figure 5.4: The hybrid branching-time hierarchy compared with the branching-time hierarchy.

On the technical side however, the techniques to prove these results have become quite a bit more involved because, compared to the proofs for the non-hybrid logics, we also need to show that the new hybrid operators cannot mimic certain temporal aspects that are added when going from one level of the hierarchy to the next.
A vital part in these proofs is to identify the right kind of structures on which these hybrid operators are limited. Because on certain kinds of structures the hybrid operators are able to use special structural characteristics to simulate certain aspects of branching-time logics while on others they cannot. For example, on finite structures hybrid operators can simulate the temporal operator $\mathsf{GF}p$ already in HCTL by simply looking for a finite cycle along which p holds. And on trees hybrid operators can even simulate some nesting of temporal operators.
Thus, identifying structures on which hybrid operators are limited in their usefulness is a vital part in proving such expressiveness results.
A natural question that arises when seeing that these results do not hold on all classes of structures is: what do these hierarchies look like on restricted classes? Especially the class of tree structures and the class of finite structures are particularly interesting.
Finite structures and tree structures are of particular interest because of their prominent use in model checking. Thus, having a good understanding



$\mathrm{HCTL}^*_{\mathsf{pp}}$
?
H_μ
$\mathrm{HCTL}^*_{\mathsf{ps}}$
$\prec$
?
$\mathrm{HCTL}^*_{\mathsf{ss}}$
$\prec$
HFCTL^+
$\equiv$
$\mathrm{HCTL} \equiv \mathrm{HCTL}^+$
$\mathrm{HCTL}^*_{\mathsf{pp}}$
?
H_μ
$\mathrm{HCTL}^*_{\mathsf{ps}}$
$\prec$
?
$\mathrm{HCTL}^*_{\mathsf{ss}}$
?
HFCTL^+
$\prec$
$\mathrm{HCTL} \equiv \mathrm{HCTL}^+$

(a) The hierarchy on finite structures (so far).

(b) The hierarchy on tree structures (so far).

Figure 5.5: The hybrid branching-time hierarchy compared with the branching-time hierarchy.

of what hybrid logics are capable of on these classes of structures may prove useful for potential applications in this area.

We have summarised the already obtained results on these two classes of structures in Figure 5.5.

5.3 The Hierarchy on Tree Structures

Theorem 5.35 shows that there are formulas in $\mathrm{HCTL}^*_{\mathsf{ps}}$ that cannot even be expressed by H_μ and since $\mathrm{HCTL}^*_{\mathsf{ss}} \prec \mathrm{H}_\mu$ this result also separates $\mathrm{HCTL}^*_{\mathsf{ss}}$ and $\mathrm{HCTL}^*_{\mathsf{ps}}$. However, this result does not transfer to the class of all trees, since the proof uses a property on cliques which inherently are not trees. Even worse, the formula that separates both logics – which states that there is an infinite path in the structure – is simply always true on a tree structure and thus does neither help to show an expressiveness gap nor the lack thereof. Thus we have to start all over again when analysing the relationship between $\mathrm{HCTL}^*_{\mathsf{ss}}$, $\mathrm{HCTL}^*_{\mathsf{ps}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}}$ on trees.

Indeed, we show in this section that the previously shown hierarchy from $\mathrm{HCTL}^*_{\mathsf{ss}}$ to $\mathrm{HCTL}^*_{\mathsf{pp}}$ *does not* carry over to the class of trees. In fact, it collapses all together. To show this, we use a result by [72] that states that Monadic Path Logic on trees coincides with an extension of CTL^* that adds a simple form of counting the number of successors.

Theorem 5.38 ([72])**.** On trees $\mathsf{Counting\text{-}CTL}^* \equiv \mathsf{MPL}$.

To show that the hierarchy collapses we simply show that the following relationships between these logics hold:

$$\mathsf{Counting\text{-}CTL}^* \preceq \text{HCTL}^*_{\mathsf{ss}} \preceq \text{HCTL}^*_{\mathsf{pp}} \preceq \mathsf{MPL}.$$

Combined with the result cited above by Moller and Rabinovich we obtain the collapse of the hierarchy.
The second relationship is obvious by the definition of $\text{HCTL}^*_{\mathsf{ss}}$ and $\text{HCTL}^*_{\mathsf{pp}}$, so we only need to show the first and third one. But before that, let us formally introduce $\mathsf{Counting\text{-}CTL}^*$ and MPL.

$\mathsf{Counting\text{-}CTL}^*$. $\mathsf{Counting\text{-}CTL}^*$ extends CTL* by adding a new type of state formula $\mathsf{D}^n\varphi$. Thus, its full grammar is given by

$$\begin{array}{rcl} \varphi & := & p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\psi \mid \mathsf{D}^n\varphi, \\ \psi & := & \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi. \end{array}$$

The semantics of state and path formulas are given exactly as for CTL* with respect to a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ and a state $s \in S$ resp. a path π in $\mathcal{K}$ and a moment on this path. The semantics of the only new operator is given as follows:

$$\begin{array}{ll} \mathcal{K}, s \models \mathsf{D}^n\varphi \text{ iff} & \text{there are at least } n \text{ different states } t \text{ with} \\ & s \to t \text{ and } \mathcal{K}, t \models \varphi. \end{array}$$

Thus, $\mathsf{Counting\text{-}CTL}^*$ – as the name suggests – can count the number of successors.

Example 5.39. The formula $\mathsf{AG}(\mathsf{D}^2\mathtt{tt} \wedge \neg\mathsf{D}^3\mathtt{tt})$ states that on all reachable states there are exactly 2 successor states.

Monadic Path Logic. Formulas of Monadic Path Logic – or short MPL– are given by the grammar

$$\varphi \ := \ x < y \mid x = y \mid p(x) \mid X(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $x, y \in \mathit{Var}$ and $X \in \mathit{Var}_2$.

They are interpreted with respect to a tree structure $\mathcal{T} = \langle S, \rightarrow, L\rangle$ and variable assignments $\sigma : \mathit{Var} \rightarrow S$, $\rho : \mathit{Var}_2 \rightarrow 2^S$ for the first- and second-order variables. The satisfaction relation is given by mutual recursion.

$$\begin{array}{lll}
\mathcal{T}, \sigma, \rho \models x < y & \text{iff} & \sigma(y) \text{ is a child of } \sigma(x), \\
\mathcal{T}, \sigma, \rho \models x = y & \text{iff} & \sigma(x) = \sigma(y), \\
\mathcal{T}, \sigma, \rho \models p(x) & \text{iff} & \sigma(x) \in L(p), \\
\mathcal{T}, \sigma, \rho \models X(x) & \text{iff} & \sigma(x) \in \rho(X), \\
\mathcal{T}, \sigma, \rho \models \neg\varphi & \text{iff} & \mathcal{T}, \sigma, \rho \not\models \varphi, \\
\mathcal{T}, \sigma, \rho \models \varphi_1 \vee \varphi_2 & \text{iff} & \mathcal{T}, \sigma, \rho \models \varphi_1 \text{ or } \mathcal{T}, \sigma, \rho \models \varphi_2, \\
\mathcal{T}, \sigma, \rho \models \exists x.\varphi & \text{iff} & \text{there is } s \in S, \text{ such that } \mathcal{T}, \sigma[x \mapsto s], \rho \models \varphi, \\
\mathcal{T}, \sigma, \rho \models \exists X.\varphi & \text{iff} & \text{there is } T \subseteq S \text{ with } T = \{s_0, s_1, s_2, \ldots\}, \\
 & & s_0 = \varepsilon \text{ and for all } i \in \mathbb{N} \text{ we have that } s_i \rightarrow s_{i+1}, \\
 & & \text{such that } \mathcal{T}, \sigma, \rho[X \mapsto T] \models \varphi.
\end{array}$$

Note that by the semantics we have restricted the second-order quantifier to only quantify over paths that start at the root of the tree.

Example 5.40. It is well-known that MPL subsumes CTL* [72]. For example the formula $\forall X.\, \forall y.\, X(y) \rightarrow \exists z.\, y < z \wedge X(z) \wedge p(z)$ states that on all paths and on all moments on this path there is a later moment that satisfies p. Hence, the MPL formula states the same property as the CTL* formula $\mathsf{AGF}p$ (if it is evaluated at the root of a tree).

Counting-CTL*, MPL and Hybrid Branching-Time Logics. We will now show the inclusions Counting-CTL* $\preceq$ $\mathrm{HCTL}^*_{\mathsf{ss}}$ and $\mathrm{HCTL}^*_{\mathsf{pp}} \preceq$ MPL. The counting operation from Counting-CTL* can easily be expressed with a number of first-order variables in MPL that are placed on different successors together with a test that all variables are placed on different states. And thus, Counting-CTL* $\preceq$ MPL. However, since we also have variables in $\mathrm{HCTL}^*_{\mathsf{ss}}$ this of course can also be done there.

Theorem 5.41. For every formula $\varphi \in$ Counting-CTL* there is an equivalent formula $\varphi' \in \mathrm{HCTL}^*_{\mathsf{ss}}$.

Proof. We give a simple translation from Counting-CTL* to $\mathrm{HCTL}^*_{\mathsf{ss}}$. We only need to translate the operator $\mathsf{D}^n\varphi$. The rest simply remains as it is. The idea is to simulate $\mathsf{D}^n\varphi$ by marking different successor states with variables. So, let $\mathsf{D}^n\varphi \in$ Counting-CTL*. We simply replace this operator by the following formula:

$$\varphi' := \downarrow x.\mathsf{EX}\downarrow x_1.\big(\tau(\varphi)\wedge$$

$$
\begin{aligned}
&@_x \mathsf{EX} \downarrow x_2.\big(\neg x_1 \wedge \tau(\varphi) \wedge \\
&\quad @_x \mathsf{EX} \downarrow x_3.\big((\neg x_1 \wedge \neg x_2) \wedge \tau(\varphi) \wedge \\
&\quad\quad \vdots \\
&\quad\quad\quad @_x \mathsf{EX} \downarrow x_{n-1}.\big((\bigwedge_{i=1}^{n-2} \neg x_i) \wedge \tau(\varphi) \wedge \\
&\quad\quad\quad\quad @_x \mathsf{EX}\big((\bigwedge_{i=1}^{n-1} \neg x_i) \wedge \tau(\varphi)\big) \ldots\big),
\end{aligned}
$$

where $\tau(\varphi)$ describes the formula φ in which all subsequent occurrences of a $\mathsf{D}^n\psi$ operator have been replaced as well.
φ' states that there are n successors of s that can successively be marked such that each state differs from the ones before and each state satisfies $\tau(\varphi)$. It should be clear that this formula is satisfied at some state s if and only if there are at least n successors of s that satisfy $\tau(\varphi)$. □

Next we show that even $\mathrm{HCTL}^*_{\mathsf{pp}}$ on trees can be embedded into MPL.

Theorem 5.42. Let $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ and let $z \in \mathit{Var}$ be a variable that does not occur in φ. Then there is a formula $\varphi' \in \mathsf{MPL}$ with a free variable z such that $\mathcal{T}, s, \sigma \models \varphi$ if and only if $\mathcal{T}, \sigma[z \mapsto s] \models \varphi'$.

Proof. To prove this we give a translation from $\mathrm{HCTL}^*_{\mathsf{pp}}$ to MPL that extends the standard translation of hybrid modal logic into first-order logic [7].
The translation of state formulas is given relative to a new variable z that simulates the "current" state in the evaluation of an $\mathrm{HCTL}^*_{\mathsf{pp}}$ variable. And the translation for path formulas is also given with respect to a set X that simulates the path.
Let $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$. We assume that the variables in φ are named $x_1, \ldots, x_k$ for some $k \in \mathbb{N}$ and that the variables used to simulate the current state and the variables occurring in φ are mutually disjoint.
The state formulas get translated via

$$
\begin{aligned}
\mathsf{tr}_z(p) &:= p(z), \\
\mathsf{tr}_z(n) &:= n(z), \\
\mathsf{tr}_z(x_i) &:= (x_i = z), \\
\mathsf{tr}_z(\neg\varphi) &:= \neg \mathsf{tr}_z(\varphi), \\
\mathsf{tr}_z(\varphi_1 \vee \varphi_2) &:= \mathsf{tr}_z(\varphi_1) \vee \mathsf{tr}_z(\varphi_2), \\
\mathsf{tr}_z(\downarrow x_i.\varphi) &:= \exists x_i.x_i = z \wedge \mathsf{tr}_z(\varphi), \\
\mathsf{tr}_z(@_{x_i} \varphi) &:= \mathsf{tr}_{x_i}(\varphi),
\end{aligned}
$$

$$\mathsf{tr}_z(\mathsf{E}\psi) \coloneqq \exists X.X(z) \wedge \mathsf{tr}_z^X(\psi)$$

and the path formulas via

$$\begin{aligned}
\mathsf{tr}_z^X(\varphi) &\coloneqq \mathsf{tr}_z(\varphi),\\
\mathsf{tr}_z^X(\neg\psi) &\coloneqq \neg\mathsf{tr}_z^X(\psi),\\
\mathsf{tr}_z^X(\psi_1 \vee \psi_2) &\coloneqq \mathsf{tr}_z^X(\psi_1) \vee \mathsf{tr}_z^X(\psi_2),\\
\mathsf{tr}_z^X(\mathsf{X}\psi) &\coloneqq \exists y.\mathsf{succ}(z,y) \wedge X(y) \wedge \mathsf{tr}_y^X(\psi),\\
\mathsf{tr}_z^X(\psi_1 \mathsf{U} \psi_2) &\coloneqq \exists y.z < y \wedge X(y) \wedge \mathsf{tr}_y^X(\psi_2) \wedge\\
&\qquad \forall x.z \leq x < y \wedge X(x) \to \mathsf{tr}_x^X(\psi_1),\\
\mathsf{tr}_z^X(\downarrow x_i.\varphi) &\coloneqq \exists x_i.x_i = z \wedge \mathsf{tr}_z^X(\varphi),\\
\mathsf{tr}_z^X(@_{x_i}\varphi) &\coloneqq \mathsf{tr}_{x_i}^X(\varphi),
\end{aligned}$$

where $\mathsf{succ}(x,y) \coloneqq x < y \wedge \neg\exists z.x < z \wedge z < y$ denotes that y is an immediate successor of x in the tree.
We now show by induction on φ that for every tree $\mathcal{T}$, every state s in $\mathcal{T}$ and every $\sigma : \mathit{Var} \to S$ it holds that $\mathcal{T}, s, \sigma \models \varphi$ if and only if $\mathcal{T}, \sigma[z \mapsto s] \models \mathsf{tr}_z(\varphi)$.
Suppose first that $\varphi = p$ and $\mathcal{T}, s, \sigma \models \varphi$. Then $s \in L(p)$ by the semantics of HCTL$^*_{\mathsf{pp}}$. By the semantics of MPL and the fact that $z \mapsto s$ we also have that $\mathcal{T}, \sigma[z \mapsto s] \models p(z)$. The same arguments show the statement for $\varphi = n$.
So, suppose that $\varphi = x_i$ for a variable x_i and $\mathcal{T}, s, \sigma \models \varphi$. This means that $\sigma(x_i) = s$ and again, with the fact that $z \mapsto s$ we get that $\mathcal{T}, \sigma[z \mapsto s] \models (x_i = z)$.
Negation and disjunction follow as usual by simple semantic arguments. The details are left out. Suppose now that $\varphi = \downarrow x_i.\varphi_1$ and that $\mathcal{T}, s, \sigma \models \varphi$. Thus, we have that $\mathcal{T}, s, \sigma[x_i \mapsto s] \models \varphi_1$. By the induction hypothesis we get that $\mathcal{T}, \sigma[x_i \mapsto s][z \mapsto s] \models \mathsf{tr}_z(\varphi_1)$. And thus, we have that $\mathcal{T}, \sigma[z \mapsto s] \models \exists x_i.x_i = s \wedge \mathsf{tr}_z(\varphi_1)$.
Now suppose that $\varphi = @_{x_i}\varphi_1$ and that $\mathcal{T}, s, \sigma \models \varphi$. This means that $\mathcal{T}, \sigma(x_i), \sigma \models \varphi_1$. By the induction hypothesis we get that $\mathcal{T}, \sigma[z \mapsto \sigma(x_i)] \models \mathsf{tr}_z(\varphi_1)$. By renaming all free occurrences of z to x_i, since z and x_i point to the same state we get that $\mathcal{T}, \sigma[z \mapsto \sigma(x_i)] \models \mathsf{tr}_{x_i}(\varphi_1)$. Since we have replaced all free occurrences of z its value is not important and we also get that $\mathcal{T}, \sigma[z \mapsto s] \models \mathsf{tr}_{x_i}(\varphi_1)$.
For the last case of $\varphi = \mathsf{E}\psi$ we first need to prove via a separate induction over path formulas that the translation holds for path formulas as well.
We prove that for all paths π, $k \in \mathbb{N}$ and $\vartheta : \mathit{Var} \to \mathbb{N}$ it holds that $\mathcal{T}, \pi, k, \vartheta \models \psi$ if and only if $\mathcal{T}, \sigma[z \to \pi^k], \rho[X \mapsto \pi] \models \mathsf{tr}_z^X(\psi)$ (with the same restriction on z as before). Note that since $\mathcal{T}$ is a tree, we can uniquely

identify a path simply by its stateset and vice versa and thus we do not distinguish between a path and its set of states and simply write $\rho(X) = \pi$ to indicate that X maps to the set of states that uniquely identifies the path π.

The base case for $\psi = \varphi$ follows from the induction hypothesis on state formulas from the outer induction. Negation, disjunction, binder and jumps follow with the same arguments as for state formulas. Note that for jumps, by the syntactic restriction of $\mathrm{HCTL}^*_{\mathsf{pp}}$ formulas, we can only jump to states previously bound on the same path thus, the step which replaces all free z variables by x_i variables still works, even for formulas of the form $X(z)$.

Suppose now that $\psi = \mathsf{X}\psi_1$ and that $\mathcal{T}, \pi, k, \vartheta \models \psi$. Thus, $\mathcal{T}, \pi, k{+}1, \vartheta \models \psi_1$. By the hypothesis we get that $\mathcal{T}, \sigma[y \to \pi^{k+1}], \rho[X \mapsto \pi] \models \mathsf{tr}^X_y(\psi_1)$ and thus also $\mathcal{T}, \sigma[z \mapsto \pi^k], \rho[X \mapsto \pi] \models \exists y.\mathsf{succ}(z, y) \wedge X(y) \wedge \mathsf{tr}^X_y(\psi_1)$ since π^{k+1} is a direct successor of π^k and π^{k+1} obviously lies in π. The last case for $\psi = \psi_1 \mathsf{U} \psi_2$ follows similarly since the translation simply directly states the semantics of the Until-operator.

To finish the outer induction, suppose that $\varphi = \mathsf{E}\psi$ and that $\mathcal{T}, s, \sigma \models \varphi$. Then there is a path π starting at s such that $\mathcal{T}, \pi, 0, \sigma \models \psi$. By the inner induction for path formulas we now get that $\mathcal{T}, \sigma[z \mapsto \pi^0], \rho[X \mapsto \pi] \models \mathsf{tr}^X_z(\psi)$ and thus $\mathcal{T}, \sigma[z \mapsto s] \models \exists X.X(z) \wedge \mathsf{tr}^X_z(\psi)$. $\square$

By combining Theorems 5.38, 5.41 and 5.42 we get that $\mathsf{Counting\text{-}CTL}^* \preceq \mathrm{HCTL}^*_{\mathsf{ss}} \preceq \mathrm{HCTL}^*_{\mathsf{pp}} \preceq \mathsf{MPL} \preceq \mathsf{Counting\text{-}CTL}^*$ and thus the collapse of the hierarchy above $\mathrm{HCTL}^*_{\mathsf{ss}}$ on trees.

Corollary 5.43. On trees we have that $\mathrm{HCTL}^*_{\mathsf{ss}} \equiv \mathrm{HCTL}^*_{\mathsf{pp}}$.

Thus, on trees the hierarchy we obtain is shaped as depicted in Figure 5.6. Thus, once again we have that the (hybrid) μ-calculus is the top logic in terms of expressiveness.

It is still open if there is an expressiveness gap between HFCTL^+ and $\mathrm{HCTL}^*_{\mathsf{ss}}$ on trees. The strategy used to separate those two logics on finite structures cannot simply be transferred to trees as it relies on paths joining up again as seen in Theorem 5.16. Also, we have seen in Example 5.19 that at least some form of nesting of temporal operators can be simulated on trees by exploiting some special structural properties of tree structures.

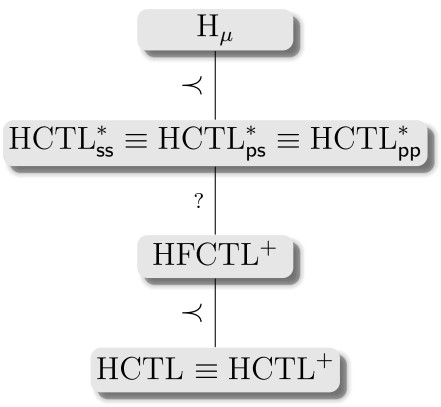

H_μ
$\prec$
$\mathrm{HCTL}^*_{\mathsf{ss}} \equiv \mathrm{HCTL}^*_{\mathsf{ps}} \equiv \mathrm{HCTL}^*_{\mathsf{pp}}$
?
HFCTL^+
$\prec$
$\mathrm{HCTL} \equiv \mathrm{HCTL}^+$

Figure 5.6: The expressiveness hierarchy on trees.

Chapter 6

Model-Checking for Hybrid Logics

One of the main tasks in formal verification is to identify with absolute certainty if a given system satisfies a specified property. This is usually referred to as model checking. A detailed analysis of the complexity of the model checking problem is thus necessary to determine possible applications for all hybrid branching-time logics defined in Chapter 3. In this chapter we thoroughly investigate their model checking problems.
The chapter is organised as follows. We begin by establishing lower bounds for the model checking complexities of the various hybrid logics. Most of these lower bounds are achieved by an encoding of a suitable tiling problem introduced as in Section 2.11. This unified framework also demonstrates how the added features of the more powerful logics can be used to encode increasingly complex problems. Afterwards we give model checking algorithms with matching complexities for each logic. In the last section of this chapter we identify some rich fragments of our hybrid logics with generally lower model checking complexities.

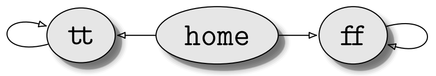
6.1 Lower bounds

6.1.1 HCTL to HCTL$^*_{\mathsf{ss}}$

We begin with the simplest hybrid branching-time logic HCTL. Remember that model checking for CTL is in P [28] which is one of the reasons why CTL is such a popular temporal logic despite its limited expressive power. Sadly, when adding hybrid operators, we get a drastic increase in model checking complexity. The proof follows the ideas of [37].

Theorem 6.1. The expression complexity for model checking HCTL is PSpace-hard.

Proof. We will show this by a reduction from the well-known QBF problem. Let $\chi := Q_1 x_1 Q_2 x_2 \ldots Q_k x_k.\, \alpha(x_1, \ldots, x_k)$ be a quantified boolean formula with $Q_i \in \{\exists, \forall\}$ and $\alpha(x_1, \ldots, x_k)$ a boolean formula in negation normal form over the variables $x_1, \ldots, x_k$. We will construct a (fixed) structure $\mathcal{K}_{\texttt{QBF}}$ and a formula φ such that $\mathcal{K}_{\texttt{QBF}}, \texttt{home} \models \varphi$ if and only if χ evaluates to $\texttt{tt}$.
The idea is to simply simulate the values true and false by two states *named* `tt` and `ff` which are reachable from a third initial state. Existential and universal quantification can then simply be simulated by placing a new variable at one or all successors of the initial state through the operators EX and AX. Evaluation of atomic formulas like x is simulated by a test if the variable is placed at the state `tt`.
For this, consider the structure $\mathcal{K}_{\texttt{QBF}}$ with the state set and nominals `home`, `tt`, `ff` interpreted as depicted in the following graph:



Now consider the HCTL formula $\varphi := \tau(\chi)$ obtained by translating χ inductively as follows:

$$\begin{aligned} \tau(\forall x.\psi(x)) &:= @_{\texttt{home}}\mathsf{AX} \downarrow x.\tau(\psi) & \tau(\exists x.\psi(x)) &:= @_{\texttt{home}}\mathsf{EX} \downarrow x.\tau(\psi) \\ \tau(\psi_1 \vee \psi_2) &:= \tau(\psi_1) \vee \tau(\psi_2) & \tau(\psi_1 \wedge \psi_2) &:= \tau(\psi_1) \wedge \tau(\psi_2) \\ \tau(\neg x) &:= @_x\texttt{ff} & \tau(x) &:= @_x\texttt{tt} \end{aligned}$$

A standard induction on χ shows that $\mathcal{K}_{\texttt{QBF}}, \texttt{home} \models \varphi$ if and only if χ evaluates to `tt`. Moreover $\mathcal{K}_{\texttt{QBF}}$ and φ can be constructed in time polynomial in $|\chi|$. □

Note that this reduction only uses hybrid operators and the X-operator and thus the resulting formula is essentially in hybrid modal logic. Thus, already hybrid modal logic possesses a PSpace-hard model checking problem.
This lower bound also transfers to all of the extensions of HCTL. Most notably the complexities of the model checking problems of HCTL$^+$, HFCTL$^+$ also rise compared to their non-hybrid variants. Note that these are known to be complete for the second level of the polynomial hierarchy Δ_2^p [67].
Model checking CTL* however was already known to be PSpace-complete [33]. We will later show that adding hybrid operators as state formulas to CTL* does not increase the complexity any further.

Corollary 6.2. The expression complexity of HCTL$^+$ as well as HFCTL$^+$ and HCTL$^*_{\mathsf{ss}}$ is PSpace-hard.

Corollary 6.3. The combined complexity for model checking HCTL as well as HCTL$^+$, HFCTL$^+$ and also HCTL$^*_{\mathsf{ss}}$ is PSPACE-hard.

This indicates that hybrid operators provide a (computationally) quite powerful mechanism even when added only to the quite limited logic CTL. For practical purposes however *data complexity* is often times a more useful indicator of how good model checking algorithms may perform. For the data complexity we can easily prove NLOGSPACE-hardness and will later see that this suffices.

Theorem 6.4. The data complexity for model checking HCTL, HCTL$^+$, HFCTL$^+$ and HCTL$^*_{\mathsf{ss}}$ is NLOGSPACE-hard.

Proof. The proof is by a simple reduction from the classical NLOGSPACE-complete problem of directed graph reachability. Since reachability of a state named `target` can be expressed by the fixed formula `EFtarget` in all of these logics. □

6.1.2 HCTL$^*_{\mathsf{ps}}$

Adding binders to path formulas increases both data- and expression complexity. We will first show that already the data complexity rises to PSPACE for HCTL$^*_{\mathsf{ps}}$ by a reduction from the n-corridor tiling problem to the model checking problem of HCTL$^*_{\mathsf{ps}}$ and subsequently prove EXPSPACE-hardness for the expression complexity of HCTL$^*_{\mathsf{ps}}$.
To show PSPACE-hardness for data complexity, let $\mathcal{T} = (T, H, V)$ be a tiling system with $T = \{t_1, \ldots, t_m\}$ and $n \in \mathbb{N}$. The idea of the reduction is quite simple. We build a structure $\mathcal{K}^n_{\mathcal{T}}$ over atomic propositions $\{\#, t_1, \ldots, t_m\}$ with n columns. Each column has a choice of all tiles from T and then continues on to the next column as depicted in Figure 6.1.
A path from the first to the last column thus gives us a tiling of a single row. The last row then points back to the first row – separated by a special state marked # which will make it easier to identify the beginning of the next row. Thus, an infinite path through this structure has the form

$$\# t_{0,0} t_{0,1} \ldots t_{0,n-1} \# t_{1,0} \ldots t_{1,n-1} \# \ldots$$

and encodes quite naturally a row by row tiling separated by #'s of the corridor of width n.

Example 6.5. Let $\mathcal{T} = (\{t_1, t_2, t_3\}, H, T$ with $H = \{(t_1, t_2), (t_2, t_3), (t_3, t_1)\})$ and $T = \{(t_1, t_3), (t_2, t_1), (t_3, t_2)\}$ be a tiling system and $n = 3$.

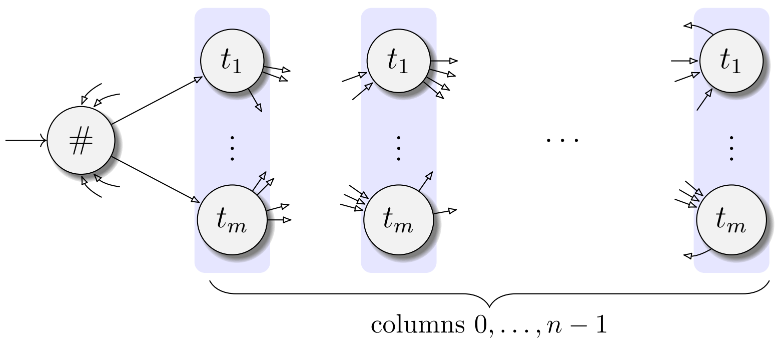
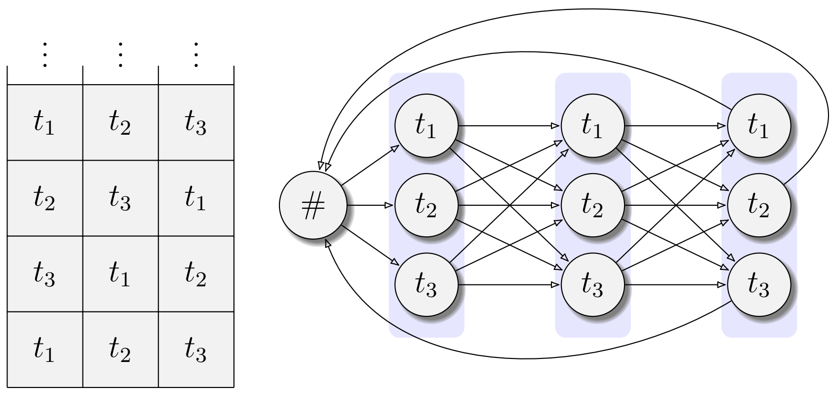


Figure 6.1: The structure $\mathcal{K}^n_\mathcal{T}$ used to encode the n-corridor tiling problem.



(a) valid $\mathcal{T}$-tiling.

(b) $\mathcal{K}^3_\mathcal{T}$

Figure 6.2: A valid $\mathcal{T}$-tiling and the structure $\mathcal{K}^3_\mathcal{T}$.

A valid $\mathcal{T}$-tiling is depicted in Figure 6.2a. The structure $\mathcal{K}^3_{\mathcal{T}}$ used in the reduction is depicted in Figure 6.2b and the path

$$\#t_1t_2t_3\#t_3t_1t_2\#t_2t_3t_1\#t_1t_2t_3\#\dots$$

through $\mathcal{K}^3_{\mathcal{T}}$ represents the valid $\mathcal{T}$-tiling depicted in Figure 6.2a.

We now construct a formula $\varphi_{\mathcal{T}}$ that holds at $\#$ if and only if there is also a *valid* tiling, i.e. a tiling that does not violate the horizontal and vertical matching relations and that also starts with t_1.
We start with the broad structure of $\varphi_{\mathcal{T}}$ which requires an infinite path that satisfies all three above mentioned conditions. Thus, the formula has the form

$$\varphi_{\mathcal{T}} := \mathsf{E}\left(\psi_{\mathsf{init}} \wedge \psi_{\mathsf{hor}} \wedge \psi_{\mathsf{vert}}\right).$$

The initial condition is easy. Starting at $\#$ we just have to check that the next state on the path is t_1. This can be checked with $\psi_{\mathsf{init}} := \mathsf{X}t_1$.
Satisfying the horizontal matching relation is also easy. We simply need to check that everytime we see some tile, the next state is a matching tile from the horizontal matching relation – or it is the last tile in a row, which can be identified with the next state being $\#$. This is checked by the following formula:

$$\psi_{\mathsf{hor}} := \mathsf{G}\left(\bigwedge_{t\in T}\left(t \to \mathsf{X}\left(\left(\bigvee_{(t,t')\in H} t'\right) \vee \#\right)\right)\right).$$

Lastly, we need to check that also the vertical matching relation is not violated along the path. To do this properly we first need a way to identify matching positions in successive rows. An easy solution would be to simply check that the n-th successor carries a tile not violating the horizontal matching relation. However, then the formula would also depend on the input n for the tiling problem which does not help in proving data complexity. This is the part where hybrid operators come into play.
Consider a path through $\mathcal{K}^n_{\mathcal{T}}$ as depicted in Figure 6.3. First, to identify a position in the *next* row we simply need to state that there is exactly one $\#$ state between now and that position. This can be done with the formula pattern $(\ldots \wedge (\neg\#\mathsf{U}(\# \wedge \mathsf{X}(\neg\#\mathsf{U}\ldots))))$ where the first dots are "now" and the second dots symbolise a position in the next row.
This however only helps to identify successive rows. To further identify matching columns we use hybrid operators. Suppose we are at i-th column – for example in state 3 in Figure 6.3 and want to identify when the path

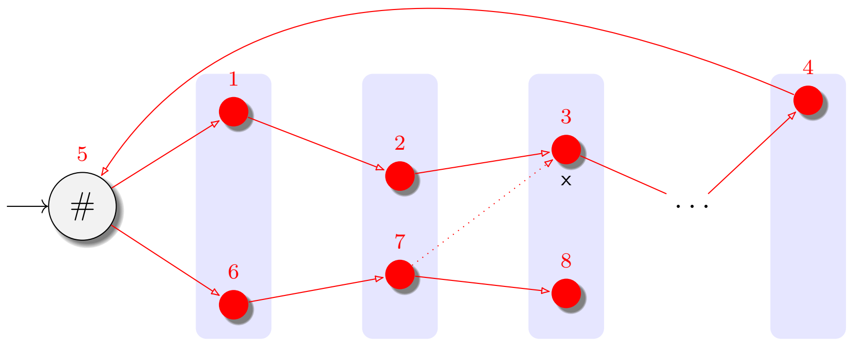


Figure 6.3: Sketch of how to identify matching columns in $\mathcal{K}^n_{\mathcal{T}}$ along a single path.

arrives at column i again – meaning state 8 in the example. To do this we *name* state 3 – say x and then continue to the next row with the pattern described above. We can now easily identify column $i-1$ by checking if there is a successor state called x. If this is the case we simply continue one step along the path and are at column i again. This idea is used in the following path formula to check the vertical matching relation at any moment on this path:

$$\psi_{\mathsf{vert}} := \mathsf{G}\left(\neg\# \to \downarrow x. \bigwedge_{t\in T} t \to \Big(\neg\#\,\mathsf{U}\,\big(\# \wedge \mathsf{X}(\neg\#\,\mathsf{U}\,(\mathsf{E}\mathsf{X}x \wedge \mathsf{X}\bigvee_{(t,t')\in V} t'))\big)\Big)\right).$$

Notice also that because of the special state # this formula also works for the first column.
This completes the reduction. The following theorems encapsulate the ideas above.

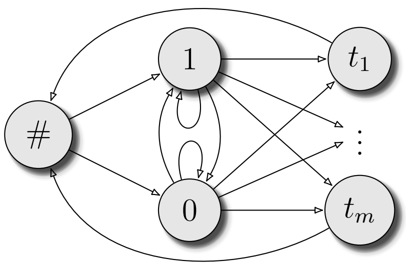
Theorem 6.6. $\mathcal{K}^n_{\mathcal{T}}, \# \models \varphi_{\mathcal{T}}$ if and only if there is a valid $\mathcal{T}$-tiling of the corridor of width n.

Theorem 6.7. The data complexity for model checking $\text{HCTL}^*_{\mathsf{ps}}$ is PSPACE-hard.

Proof. First, we observe that in the reduction above only the structure $\mathcal{K}^n_{\mathcal{T}}$ depends on n and $\varphi_{\mathcal{T}}$ only depends on the tiling system. Clearly both $\mathcal{K}^n_{\mathcal{T}}$ and $\varphi_{\mathcal{T}}$ can be constructed in polynomial time.
Let $\mathcal{T}_0$ be the tiling system for $k = 0$ from Proposition 2.64. With Theorem 6.6 we get that $\mathcal{K}^n_{\mathcal{T}_0}, \# \models \varphi_{\mathcal{T}_0}$ if and only if the n-corridor $\mathcal{T}_0$-tiling

problem has a solution. Thus, since only the structure depends on n we get PSpace-hardness for the data complexity of $\mathrm{HCTL}^*_{\mathsf{ps}}$, in fact even for $\mathrm{H}^1\mathrm{CTL}^*_{\mathsf{ps}}$ already. $\square$

The expression complexity of $\mathrm{HCTL}^*_{\mathsf{ps}}$ is even higher. We will prove ExpSpace-hardness for this problem by a reduction from the 2^n-corridor tiling problem. Contrary to the previous reduction however we now need a fixed structure and, despite the exponentially larger width of the corridor, a still polynomially sized formula that encodes this problem.
The basic idea to encode a valid tiling along a single path stays the same, however we introduce an additional concept to keep the formula small enough: a counter. Every tile encoded by a state along the path will be preceeded by a binary enumeration of its column – encoded via a sequence of n bits by states marked 0 and 1.
The proof is by a reduction from the 2^n-corridor tiling problem. Consider the structure $\mathcal{K}_\mathcal{T}$



over the atomic propositions $\{\#, 0, 1, t_1, \dots, t_m\}$ as depicted above. We will also refer to the states using their unique propositions as names. There are edges from # to both 0 and 1 and from both values there are self-loops as well as transitions to each other and to all the tiles t_i. Additionally, from each tile t_i there are edges back to #.
Paths starting at # followed by n bits 0 or 1 and a tile t_i encode a single cell of a potential tiling. The state # just marks the beginning of the encoding of a single cell. The n bits encode the column of the cell in binary and the tile t_i describes the tile that is placed in this cell. A path repeating this pattern 2^n times with a counter that starts with $0\,0 \dots 0$ and increases by 1 with each repetition thus encodes a single row of a potential tiling. An infinite path repeating the pattern for a single row of a tiling then encodes a $\mathcal{T}$-tiling of the 2^n-corridor.
We will construct an $\mathrm{HCTL}^*_{\mathsf{ps}}$ formula $\varphi^n_\mathcal{T}$ which holds at # if there is a path of the form

$$\#\,0\,0\dots0\,t_{0,0}\,\#\,1\,0\dots0\,t_{0,1}\,\#\dots\#\,1\,1\,\dots\,1\,t_{0,2^n-1}\,\#\dots$$

that encodes a *valid* $\mathcal{T}$-tiling.
The basic structure of the formula will be the same as before with the addition of a part that requires the counter to behave correctly. Thus, we have

$$\varphi_{\mathcal{T}}^{n} \ := \ \mathsf{E}\,(\psi_{\mathsf{init}} \wedge \psi_{\mathsf{count}} \wedge \psi_{\mathsf{hor}} \wedge \psi_{\mathsf{vert}})\,.$$

However each formula also needs to take into account the slightly more complex encoding.
ψ_{init} not only states that we begin with tile t_1 but also with the counter set to 0 and sets up the general structure of # followed by n bits of 0 or 1 followed by some tile:

$$\psi_{\mathsf{init}} \ := \ \mathsf{X}\bigwedge_{i=0}^{n-1}\mathsf{X}^i 0 \wedge \mathsf{X}^{n+1}t_1 \wedge \mathsf{G}\left(\# \to \mathsf{X}\bigwedge_{i=0}^{n-1}\mathsf{X}^i(0 \vee 1) \wedge \mathsf{X}^{n+1}\bigvee_{j=1}^{m} t_j\right).$$

The next formula states that the counter behaves correctly, i.e. it increases by one with each repetition. To understand this formula it is helpful to remember how binary counters work. Let $b := b_0b_1\dots b_{n-1}$ and $b' := b'_0b'_1\dots b'_{n-1}$ be two binary counter values. b' is the direct successor of b if and only if the following two conditions hold:

- $b_0 \neq b'_0$, i.e. the least significant bit changes, and
- for all $i = 1,\dots,n-1$ it holds that $b_i = b'_i$ if and only if $b_{i-1} \leq b'_{i-1}$, i.e. the bits at position i only increase if and only if the previous bit has been flipped from 1 to 0.

The former condition can be formalised by the propositional formula $b_0 \leftrightarrow \neg b'_0$ and the latter by the formula $(b_i \leftrightarrow b'_i) \leftrightarrow (b_{i-1} \to b'_{i-1})$. Notice that this formula also holds for the case of $b = 1\,1\dots1$ and $b' = 0\,0\dots0$.
With this we can build ψ_{count} by checking that successive countervalues after each # behave in exactly this way:

$$\begin{aligned}\psi_{\mathsf{count}} \ := \ \mathsf{G}\Big(\# \to & \mathsf{X}(0 \leftrightarrow \mathsf{X}^{n+2}1) \wedge \\ & \bigwedge_{i=1}^{n-1}((\mathsf{X}^i 1 \leftrightarrow \mathsf{X}^{n+2+i}1) \leftrightarrow (\mathsf{X}^{i-1}1 \to \mathsf{X}^{n+1+i}1))\Big).\end{aligned}$$

Notice that by ψ_{init} the n bits after # are forced to be either 0 or 1 so this formula only works properly in combination with ψ_{init}. The formula also resets the counter properly, i.e. if the counter is at $1\,1\dots1$, the next repetition correctly resets to $0\,0\dots0$ creating an infinite loop of increasing counter values from 0 to $2^n - 1$ along the path.

The next formula simply checks the horizontal matching relation. For every position on the path we check that if it is a tile then it is either the rightmost tile in a row – indicated by the next counter being $0\,0\dots 0$ – or the tile in the next cell does not violate the horizontal matching relation.

$$\psi_{\mathsf{hor}} \;:=\; \mathsf{G}\left(\bigwedge_{t\in T} t \to \mathsf{X}^2\Big((\bigwedge_{i=0}^{n-1}\mathsf{X}^i 0) \vee \mathsf{X}^n \bigvee_{(t,t')\in H} t'\Big)\right).$$

The last thing is to check the vertical matching relation. As before at this point the usual temporal operators do not suffice anymore and hybrid operators come into play. We cannot simply check with the original temporal operators that tiles placed on top of each other match since the "distance" along the path between these two positions is too big. However, we can also not use some neat property of the structure in question as before since the structure encodes basically no information at all about the sequence of tiles along the path.
So the idea is to find corresponding counter positions. For each tile we use n variables to "store" the countervalue and then use them to find the *immediate* next position where this exact sequence of n bits is seen again. Thus, the vertical matching is checked by

$$\psi_{\mathsf{vert}} \;:=\; \mathsf{G}\Bigg(\# \to \mathsf{X}\downarrow x_1.\mathsf{X}\downarrow x_2.\dots\mathsf{X}\downarrow x_n.\mathsf{X}\downarrow x. \\ \widehat{\mathsf{F}}\Big(\# \wedge \bigwedge_{i=1}^{n}(\mathsf{X}^i 0 \leftrightarrow @_{x_i}0) \wedge \bigvee_{(t,t')\in V}(\mathsf{X}^{n+1}t' \wedge @_x t)\Big)\Bigg)$$

with

$$\widehat{F}(\psi) \;:=\; \neg\varphi_p \mathsf{U}\psi$$

and

$$\varphi_p \;:=\; \bigwedge_{i=1}^{n}\bigwedge_{p\in\{0,1\}}(X^i p \leftrightarrow @_{x_i}p).$$

This completes the reduction.

Theorem 6.8. $\mathcal{K}_{\mathcal{T}}, \# \models \varphi^n_{\mathcal{T}}$ if and only if there is a valid $\mathcal{T}$-tiling of the 2^n-corridor tiling problem.

To summarise, we get the following two results for expression and combined complexity of the model checking problem for $\mathrm{HCTL}^*_{\mathsf{ps}}$.

Theorem 6.9. The expression complexity for model checking $\mathrm{HCTL}^*_{\mathsf{ps}}$ is ExpSpace-hard.

Proof. It is clear that in the reduction above both $\mathcal{K}_{\mathcal{T}}$ and $\varphi^n_{\mathcal{T}}$ can be constructed in polynomial time.
Since $\mathcal{K}_{\mathcal{T}}$ does not depend on n and by Proposition 2.64 there are fixed tiling systems for which the 2^n-corridor tiling problem is ExpSpace-hard we obtain ExpSpace-hardness for the expression complexity of $\mathrm{HCTL}^*_{\mathsf{ps}}$ model checking. □

Corollary 6.10. The combined complexity for the model checking problem of $\mathrm{HCTL}^*_{\mathsf{ps}}$ is ExpSpace-hard.

6.1.3 $\mathrm{HCTL}^*_{\mathsf{pp}}$

$\mathrm{HCTL}^*_{\mathsf{pp}}$ now also allows jumps on path formulas – at least if the referenced variable was bound on the same path.
The lower bound for the data complexity for this logic simply transfers upwards from its syntactic fragment $\mathrm{HCTL}^*_{\mathsf{ps}}$. We will see in the next section that this lower bound is tight.

Corollary 6.11. The data complexity for the model checking problem of $\mathrm{HCTL}^*_{\mathsf{pp}}$ is PSpace-hard.

In contrast to the data complexity, the expression complexity of the model checking problem rises dramatically. In the following we will prove hardness for m-ExpSpace for each $m \geq 0$.
We will show this by reductions from the 2^n_m-corridor tiling problem to the model checking problem of $\mathrm{HCTL}^*_{\mathsf{pp}}$ for each $m \in \mathbb{N}$. The case for 1-ExpSpace has already been done in the previous section. The basic idea of tiles marked by a counter value indicating the column stays the same, however we need a way to massively scale the counter values before each tile.
First, let us fix a tiling system $\mathcal{T} = (T, H, V)$ with $T = \{t_1, \dots, t_k\}$ and $n \in \mathbb{N}$ for the remainder of the section. And also, let $m \in \mathbb{N}$ and $Prop_m := \{\$_i, 0_i, 1_i \mid i = 1, \dots, m\}$ be a set of atomic propositions.
We first introduce "big counters" that encode the numbers $0, \dots 2^n_m - 1$ in a uniform manner along the lines of [79].

Definition 6.12. Let $|.|_m : \{0_m, 1_m\}^* \to \{0,1\}^*$ be the homomorphisms defined through $|0_m| = 0$ and $|1_m| = 1$ for all $m \in \mathbb{N}$. And let $bin_m : \mathbb{N} \to$

$\{0,1\}^*$ for $m \geq 1$ be the functions that map a natural number to its binary representation potentially filled up to at least 2^n_{m-1} digits.
For $i \in \{0, \ldots, 2^n - 1\}$ we define

$$c_{1,i} \;\; \coloneqq \;\; \$_1 b_1^1 b_1^2 \ldots b_1^n$$

with $b_1^1, \ldots, b_1^n \in \{0_1, 1_1\}$ such that $|b_1^1 b_1^2 \ldots b_1^n|_1 = bin_1(i)$.
And for $m > 1$ and $i \in \{0, \ldots 2^n_m - 1\}$ we define

$$c_{m,i} \;\; \coloneqq \;\; \$_m b_m^1 c_{m-1,0} b_m^2 c_{m-1,1} \ldots b_m^{2^n_{m-1}} c_{m-1,2^n_{m-1}-1}$$

with $b_m^1, \ldots, b_m^{2^n_{m-1}} \in \{0_m, 1_m\}$ such that $|b_m^1 b_m^2 \ldots b_m^{2^n_{m-1}}|_m = bin_m(i)$.

Let $\mathcal{K}^m_{\mathcal{T}} = \langle S, \rightarrow, L \rangle$ be the complete clique over the state set $S = T \cup \{\#\} \cup$ $Prop_m$ and the same atomic propositions such that $L(x) = \{x\}$ for every $x \in S$. Thus, each state has a transition to all states, including itself and every state is labeled only by its name.
We will construct HCTL$^*_{\mathsf{pp}}$ formulas $\varphi_{\mathcal{T},n,m}$ for each $m \in \mathbb{N}$ such that $\varphi_{\mathcal{T},n,m}$ holds at $\#$ on $\mathcal{K}^m_{\mathcal{T}}$ if and only if there is a path of the form

$$\# t_{0,0} c_{m,0} t_{0,1} c_{m,1} \ldots t_{0,2^n_m-1} c_{m,2^n_m-1} \# t_{1,0} c_{m,0} \ldots$$

that encodes a *valid* $\mathcal{T}$-tiling. We use $\#$ as a special state to indicate that a new row begins followed by tiles t_i and large counters $c_{m,j}$ indicating the position of the tile in the row.
Before we start to build $\varphi_{\mathcal{T},n,m}$ we need to do some groundwork. As a general abbreviation we use

$$\Sigma_{>i} \;\; \coloneqq \;\; \bigvee_{k=i+1}^{m} (0_i \vee 1_i \vee \$_i) \vee \# \vee \bigvee_{t \in T} t$$

to talk about any atomic proposition indexed by at least $i+1$. For our purposes tiles and $\#$ are always considered to be of highest possible index.
And we use the path formula

$$\widehat{\mathsf{F}}_m \psi \;\; \coloneqq \;\; (\neg \$_m) \mathsf{U} \psi$$

to state that there is some future position between now and the beginning of the next level-m counter that satisfies ψ.
The next formulas state basic properties of the counter. We begin with the level-m counter set to zero. For this we use the formula

$$\mathsf{init}_m \;\; \coloneqq \;\; \$_m \wedge \mathsf{X}((\neg 1_m) \mathsf{U} \$_m)$$

to state that there is no 1_m between the beginning of this counter and the beginning of the next level m counter.
The next set of formulas helps us to identify positions that encode the same level-m counter. They are built inductively over m with a free variable x and evaluate to true if and only if the current position and the position that x refers to are each at the beginning of a counter that encodes the same value as the other one. For $m = 1$ this is quite easy to check:

$$\mathsf{Eq}_1(x) \;\coloneqq\; \mathsf{X} \bigwedge_{i=0}^{n-1} (@_x \mathsf{X}^i 1_1 \leftrightarrow \mathsf{X}^i 1_1).$$

Eq_{m+1} then checks that there are no two points x, y in their respective level $m+1$ blocks such that x, y represent the same counting position – identified through the beginning of the same level m countervalue – but disagree in their own bit:

$$\mathsf{Eq}_{m+1}(x) \;\coloneqq\; \neg\Big(\mathsf{X}\widehat{\mathsf{F}}_{m+1} \downarrow y.\, @_x \mathsf{X}\widehat{\mathsf{F}}_{m+1}(\mathsf{X} \downarrow x.\, @_y \mathsf{X}\mathsf{Eq}_m(x)) \\ \wedge \neg\,(1_{m+1} \leftrightarrow @_y\, 1_{m+1})\Big).$$

Finally, we also want to identify successive counting position. The formulas Next_m are also built inductively and only evaluate to true if the current position is at the beginning of a sequence of states that encodes the direct successor of the countervalue starting at position x. Again, the formula for $m = 1$ is quite easy simply stating how to increase a binary counter similar to how it is done for the lower bound of $\mathrm{HCTL}^*_{\mathsf{ps}}$ model checking in Section 6.1.2:

$$\mathsf{Next}_1(x) \;\coloneqq\; (@_x \mathsf{X}1_1 \leftrightarrow \mathsf{X}\neg 1_1) \wedge \\ \bigwedge_{i=2}^{n} \left((\mathsf{X}^i 1_1 \leftrightarrow (@_x \mathsf{X}^i 1_1)) \leftrightarrow ((@_x \mathsf{X}^{i-1} 1_1) \to \mathsf{X}^{i-1} 1_1)\right).$$

And finally for Next_{m+1} we state similarly to Eq_m that there are no two positions x, y representing the same counting position that do not correctly increase the counter:

$$\mathsf{Next}_{m+1}(x) \;\coloneqq\; (@_x \mathsf{X}1_{m+1} \leftrightarrow \mathsf{X}\neg 1_{m+1}) \wedge \\ \neg\Big[\mathsf{X}\widehat{\mathsf{F}}_{m+1} \downarrow y.\, @_x \mathsf{X}\widehat{\mathsf{F}}_{m+1} \downarrow x. \Big(\mathsf{Eq}_m(y) \wedge \\ \neg\Big[\Big((@_y \widehat{\mathsf{F}}_{m+1}(\mathsf{X}\mathsf{Next}_m(y) \wedge 1_{m+1})) \leftrightarrow (@_x \widehat{\mathsf{F}}_{m+1}(\mathsf{X}\mathsf{Next}_m(x) \wedge 1_{m+1}))\Big) \\ \leftrightarrow \big(1_{m+1} \to (@_y\, 1_{m+1})\big)\Big]\Big)\Big].$$

With this we have enough basics to start building $\varphi_{\mathcal{T},n,m}$. Again, the basic structure of the formula stays mostly the same as in the previous reductions. Thus we get

$$\varphi_{\mathcal{T},n,m} \;\coloneqq\; \mathsf{E}\,(\psi_{\mathsf{init}} \wedge \psi^m_{\mathsf{count}} \wedge \psi_{\mathsf{hor}} \wedge \psi_{\mathsf{vert}})$$

where ψ_{init} will check the initial condition of the tiling problem and some minor conditions for the encoding, ψ^m_{count} will check that all counters up to level m behave correctly and $\psi_{\mathsf{hor}}, \psi_{\mathsf{vert}}$ will again check that horizontal and vertical matching relations are not violated along the path.
We will start by building ψ_{init} which encodes that we begin with # followed by the initial tile t_1. Also, it sets up the basic structure of the path: Two steps after each # the level-m counter gets (re)initialised and before each level-m counter, there needs to be exactly one tile:

$$\psi_{\mathsf{init}} \;\coloneqq\; \# \wedge \mathsf{X}t_1 \wedge \mathsf{G}(\# \leftrightarrow \mathsf{XXinit}_m) \wedge \mathsf{G}\left(\left(\bigvee_{t\in T} t \wedge \bigwedge_{t'\neq t} \neg t'\right) \leftrightarrow \mathsf{X}\$_m\right).$$

The formula ψ^m_{count} sets up the correct behaviour of the (already initialised) counters. It is built recursively. We start with $m = 1$:

$$\psi^1_{\mathsf{count}} \;\coloneqq\; (\Sigma_{>1}\mathsf{U}\$_1) \wedge \mathsf{G}\big(\$_1 \rightarrow \mathsf{X}{\downarrow}x.\mathsf{X}^n(\Sigma_{>1}\,\mathsf{U}\,(\$_1 \wedge \mathsf{Next}_1(x)))\big).$$

$\psi^{m+1}_{\mathsf{count}}$ then recursively states that the level-m counter behave correctly and two steps after each $\$_{m+1}$ a level-m counter begins set to zero and each level-m counter is preceeded by exactly one level-$(m+1)$ bit and these bits count up correctly:

$$\begin{aligned}\psi^{m+1}_{\mathsf{count}} \;\coloneqq\; & \psi_m \wedge (\Sigma_{>m+1}\mathsf{U}\$_{m+1}) \wedge \mathsf{G}\big(\$_{m+1} \rightarrow \mathsf{XXinit}_m\big) \wedge \\ & \mathsf{G}\big((0_{m+1} \vee 1_{m+1}) \rightarrow \mathsf{X}\$_m\big) \wedge \\ & \mathsf{G}\big(\$_{m+1} \rightarrow {\downarrow}x.\,((\Sigma_{<m} \vee 1_m)\mathsf{U}\Sigma_{>m+1}) \vee \\ & \qquad ((\Sigma_{<m} \vee 0_m \vee 1_m)\mathsf{U}\,(\Sigma_{>m+1}\mathsf{U}(\$_{m+1} \wedge \mathsf{Next}_{m+1}(x))))\,\big).\end{aligned}$$

To finish the reduction,

$$\psi_{\mathsf{hor}} \;\coloneqq\; \mathsf{G}\left(\bigwedge_{t\in T} t \rightarrow (\neg \bigvee_{t\in T} t)\mathsf{U}(\bigvee_{(t,t')\in H} t' \vee \#)\right)$$

and

$$\psi_{\mathsf{vert}} \;\coloneqq\; \mathsf{G}\left(\bigwedge_{t\in \mathcal{T}} t \rightarrow {\downarrow}x.(\neg\#)\mathsf{U}(\# \wedge (\neg\#)\mathsf{U}(\mathsf{Eq}_m(x) \wedge \bigvee_{(t,t')\in V} t'))\right)$$

require that the horizontal and vertical matching relations are fulfilled.
The size of these formulas is easily seen to be only polynomial in n, m.

Theorem 6.13. $\mathcal{K}^m_{\mathcal{T}}, \# \models \varphi_{\mathcal{T},n,m}$ if and only if there is a valid $\mathcal{T}$-tiling of the 2^n_m-corridor.

Moreover, $\mathcal{K}^m_{\mathcal{T}}$ does not depend on n. Thus we get the following result.

Theorem 6.14. The expression complexity for model checking $\mathrm{HCTL}^*_{\mathsf{pp}}$ is hard for m-ExpSpace for any $m \geq 0$.

6.1.4 The Fully Hybrid μ-calculus H_μ

We will now show that H_μ has an ExpTime-hard model checking problem. We will do this by a reduction from the n-corridor tiling game. Remember that in a tiling game two players – Adam and Eve – play against each other. To win, it is Eve's task to tile the entire corridor apart from the first tile in each row being picked by Adam first.
Again, we fix a tiling system $\mathcal{T} = (T, H, V)$ with tiles $T = \{t_1, \ldots, t_k\}$. We build a structure $\mathcal{K}_{\mathcal{T}}$ and a formula $\varphi^n_{\mathcal{T}}$ such that $\mathcal{K}_{\mathcal{T}} \models \varphi^n_{\mathcal{T}}$ if and only if Eve has a winning strategy in the n-corridor tiling game.
Let $\mathcal{K}_{\mathcal{T}} = \langle S, \to, L\rangle$ be the Kripke structure over $\mathit{Prop} = \{t_1, \ldots, t_k, \mathsf{last}\}$ with $S = T \cup \{t'_1, \ldots, t'_k\}$, $L(t_i) = \{t_i, t'_i\}$ and $L(\mathsf{last}) = \{t'_1, \ldots, t'_k\}$. Further, the transition relation for $t_i, t_j \in T$ and $t'_i, t'_j \in \{t'_1, \ldots, t'_k\}$ is given by the following equivalences:

$$\begin{aligned}
&t_i \to t_j \text{ iff } (t_i, t_j) \in H,\\
&t_i \to t'_j \text{ iff } (t_i, t_j) \in H,\\
&t'_i \to t_j \text{ for all } i, j \in \{1, \ldots, k\}.
\end{aligned}$$

Thus, $\mathcal{K}_{\mathcal{T}}$ has two copies of tiles. In the first one, we encode the horizontal matching relation from the tiling system, meaning that we only can change states from one tile to another (even to the other copy of tiles) iff both states do not violate H. The second copy will be used to indicate the last state of a row after which a new row begins that does not have to satisfy any horizontal matching relations which means any tile can be chosen as long as it satisfies the vertical matching relation. This will be checked in the formula $\varphi^n_{\mathcal{T}}$.

Example 6.15. Let $\mathcal{T}$ be the tiling system from Example 6.5. The structure $\mathcal{K}_{\mathcal{T}}$ which is used in the reduction is depicted in Figure 6.4. The two copies of the tiling system are each marked in blue. The black transitions are simply an encoding of the horizontal matching relation and the possibility to change into the second copy which will mark the last tiles in a row. The red transitions then open up all possibilities for the initial tile in the next row.

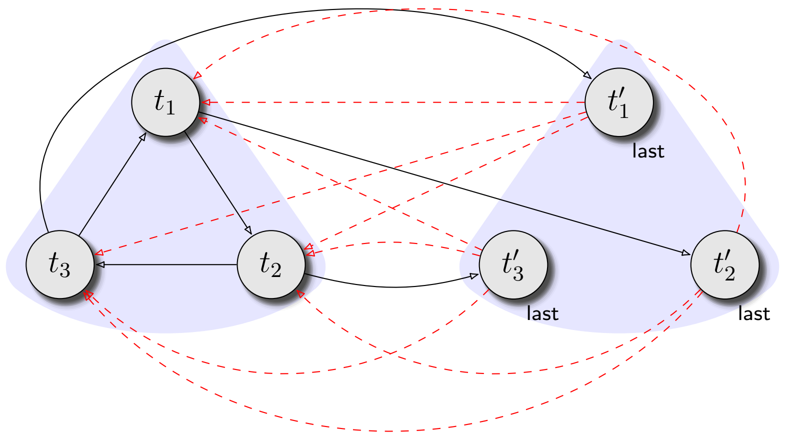


Figure 6.4: The Kripke structure $\mathcal{K}_\mathcal{T}$ used in the reduction from the n-corridor tiling game to the model checking problem for H_μ.

The formula $\varphi^n_\mathcal{T}$ then needs to encode the evolution of the tiling game. Since the game continues forever if Eve can win the game, we employ a greatest fixpoint to model this behaviour. We use n variables to "store" the previous row and to check the vertical matching relation. Moreover, the formula is written from Eve's perspective. Thus, finding a suitable tile for the next position is identified by a $\Diamond$ while Adam's picks for the first tile of a row are done using a $\Box$ to indicate that, no matter which tile Adam chooses, Eve has to find a valid answer to his choice. We use the proposition last to control in which part of the structure the tile is chosen, i.e. to control the transition relation for the next tile.
We use

$$\begin{aligned} vm_i &:= \bigvee_{(t,t')\in V} t' \wedge @_{x_i} t \wedge \neg \mathsf{last}, \\ vm_i^{\mathsf{last}} &:= \bigvee_{(t,t')\in V} t' \wedge @_{x_i} t \wedge \mathsf{last} \end{aligned}$$

as shorthands to check the vertical matching relation between the current state and x_i. With this we can define

$$\begin{aligned} \varphi^n_\mathcal{T} := {} & {\downarrow} x_0.\Diamond {\downarrow} x_1.\Diamond {\downarrow} x_2.\ldots\Diamond {\downarrow} x_{n-1}.\Box\Big(\nu Y.vm_0 \rightarrow \\ & {\downarrow} x_0.\Diamond(vm_1 \wedge {\downarrow} x_1.\Diamond(vm_2 \wedge {\downarrow} x_2.\Diamond(\ldots\Diamond(vm^{\mathsf{last}}_{n-1} \wedge {\downarrow} x_{n-1}.\Box Y)\ldots)))\Big). \end{aligned}$$

The following theorem proves correctness of the reduction. To prove this we utilise the model checking games for H_μ and associate winning strategies in the model checking game to strategies in the tiling game and vice versa.

Theorem 6.16. $\mathcal{K}_\mathcal{T}, t_1 \models \varphi^n_\mathcal{T}$ if and only if Eve has a winning strategy in the n-corridor tiling game.

Proof. Suppose that $\mathcal{K}_\mathcal{T}, t_1 \models \varphi^n_\mathcal{T}$. By Theorem 3.20 player $\boldsymbol{V}$ has a winning strategy in the model checking game $\mathcal{G}(\mathcal{K}, t_1, \sigma, \varphi^n_\mathcal{T})$ for some $\sigma : \mathit{Var} \to S$. Thus, for each $\Diamond$ in $\varphi^n_\mathcal{T}$ the winning strategy for $\boldsymbol{V}$ can choose a suitable tile. The rest of the game is either deterministic or is chosen by $\boldsymbol{R}$ – especially to control whether the tiles satisfy the vertical matching relation and the tile for the new row after n steps. The choices of tiles by $\boldsymbol{V}$ can immediately be translated into a winning strategy for Eve in the tiling game. Since all plays that are played according to this strategy are winning for $\boldsymbol{V}$, this strategy translates to a winning strategy for Eve that eventually produces a valid $\mathcal{T}$-tiling.
Suppose now that Eve has a winning strategy in the tiling game. As before these choices can immediately be translated into a winning strategy in the model checking game. Plays according to this strategy are easily seen to be won by $\boldsymbol{V}$. Infinite games are won because the only fixed point is of type ν and finite games are won by $\boldsymbol{V}$ because the tiles chosen by Eve actually satisfy the vertical matching relation or the tile chosen by Adam violates it. $\square$

Using Proposition 2.67 we get that there is a fixed tiling $\mathcal{T}_0$ for which the n-corridor $\mathcal{T}_0$-tiling game is ExpTime-hard. If we put this together with the observation that the structure $\mathcal{K}_\mathcal{T}$ used in the reduction does not depend on n, we obtain the following result about the model checking complexity for H_μ.

Theorem 6.17. The expression complexity – and thus also the combined complexity – of the model checking problem for H_μ is ExpTime-hard.

The data complexity on the other hand is much simpler. We only show P-hardness here and give a matching upper bound in Section 6.2.4.

Theorem 6.18. The data complexity of H_μ is P-hard.

Proof. We can see this by a reduction from the alternating graph reachability problem to the model checking problem of H_μ (in fact even L_μ).
The alternating graph reachability problem is the following: Given are a directed graph and a partition of its nodes into subsets A and B as well as

a starting state s and a target state t. Players 1 and 2 now play a game starting with a pebble placed on s in which they move this pebble across the graph. If the current state u on which the pebble lies belongs to A then player 1 can choose a successor and move the pebble there and if $u \in B$ then player 2 can choose a successor. Player 1 wins the game if and only if the state t is reached. It is well-known that this problem is P-hard [21].
We can model the directed graph as a Kripke structure $\mathcal{K}$ and the partitioning of its states by two atomic propositions a and b. A state v is labeled by a if and only if $v \in A$ and similarly for b. We also label the target state with a proposition t. Let $\varphi := \mu X.t \vee (a \wedge \Diamond X) \vee (b \wedge \Box X)$. It is not hard to see that $\mathcal{K}, s \models \varphi$ if and only if player 1 has a winning strategy, i.e. can force the game to reach t. Moreover, the formula does not depend on the input. $\Box$

6.2 Upper Bounds

In the following section we will present model checking algorithms for each logic that match the lower bounds presented in the previous section.

6.2.1 From HCTL to HCTL$^*_{\mathsf{ss}}$

Remember that already HCTL model checking is PSPACE-hard. However, we will now show that the complexity does not increase further up to HCTL$^*_{\mathsf{ss}}$, by giving a model checking algorithm for HCTL$^*_{\mathsf{ss}}$ that works in polynomial space.
To understand this procedure it is helpful to recall how CTL* model checking in PSPACE can be achieved. The standard argument for this is the following [33]:

1. Rewrite φ using the equivalence $\mathsf{E}\psi \equiv \neg\mathsf{A}\neg\psi$ such that no existential path quantifiers in φ remain.

2. Take φ and recursively evaluate all of its maximal subformulas of the form $\mathsf{A}\psi$. If ψ does not contain any subformulas of this form then ψ is a pure linear-time formula which can be evaluated using an LTL model checking procedure. Otherwise, replace all subformulas of the form $\mathsf{A}\psi$ by a new proposition p_ψ and mark all states in $\mathcal{K}$ that satisfy $\mathsf{A}\psi$ by p_ψ.

The space needed for this procedure is dominated by the space needed by the LTL model checking algorithm. This problem is known to be in PSPACE [84]. Additionally, we need space of size at most $|\varphi| \cdot |\mathcal{K}|$ to label the structure

Algorithm 1 Model checking HCTL$^*_{\mathsf{ss}}$.

```
procedure $\mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi, \sigma)$                    ▷ $\mathcal{K} = \langle S, \to, L\rangle$
    case $\varphi$ of
    $p$:  $\{s \in S \mid s \in L(p)\}$
    $x$:  $\{\sigma(x)\}$
    $\neg\varphi'$:  $S \setminus \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma)$
    $\varphi' \vee \varphi''$:  $\mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma) \cup \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi'', \sigma)$
    $\downarrow x.\varphi'$:  $\{s \mid s \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma[x \mapsto s])$
    $@_x\,\varphi'$:  if $\sigma(x) \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma)$ then $S$ else $\emptyset$
    $\mathsf{A}\psi$:
        let $\varphi_1, \ldots, \varphi_m$ be maximal state subformulas of $\psi$
        let $p_1, \ldots, p_m$ be new atomic propositions
        for $i = 1, \ldots, m$ do
            $L := L[p_i \mapsto \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi_i, \sigma)]$
        end for
        $\psi' \leftarrow \varphi[p_1/\varphi_1, \ldots, p_m/\varphi_m]$
        $L \leftarrow L \cup \sigma$
        LTL-MC$(\mathcal{K}, \psi')$
    end case
end procedure
```

with the potentially new atomic propositions. Thus, CTL* model checking can also be done in PSpace.

Algorithm 1 now realises a global model checking procedure for HCTL$^*_{\mathsf{ss}}$ inspired by this approach. It takes as arguments a Kripke structure $\mathcal{K}$, an HCTL$^*_{\mathsf{ss}}$ state formula φ and a variable assignment σ that maps variables to states in $\mathcal{K}$. For closed formulas φ the variable assignment σ may initially be empty.

The idea of this model checking procedure is very similar to that of CTL*. However, a small conceptual extension compared to CTL* is needed. In the process of evaluating hybrid formulas we need to deal with *open* formulas, i.e. formulas with free variables.

Take for example the formula $\mathsf{AG}\downarrow x.\neg\mathsf{A}\neg\mathsf{XF}x$, stating that on any path at any given moment there is a non-trivial cycle. To evaluate the formula at some point we need to evaluate the formula $\mathsf{A}\neg\mathsf{XF}x$ which is not closed and thus cannot simply be evaluated and replaced by a fresh proposition independently but needs to take the value of the variable x into account. To maintain the usual bottom-up algorithm for CTL* we could evaluate $\mathsf{A}\neg\mathsf{XF}x$ with respect to every possible value of x and mark the states accordingly. This however would clutter the labeling of the structure massively.

Instead, we use a top-down approach to find values for the variables first before evaluating the subformulas. In the worst case this still means that each subformula may be evaluated for each possible values of its free variables.

Theorem 6.19. For all Kripke structures $\mathcal{K} = \langle S, \to, L\rangle$, $s \in S$ and variable assignments $\sigma : \mathit{Var} \to S$ it holds that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $s \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi, \sigma)$.

Proof. We show this by an induction on φ. So, let $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure, $s \in S$, $\sigma : \mathit{Var} \to S$ and $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$. For the base cases we immediately get the result:

$\varphi = p$: An inspection of the semantics of $\mathrm{HCTL}^*_{\mathsf{ss}}$ and the first case of the algorithm gives us $\mathcal{K}, s, \sigma \models \varphi \Leftrightarrow s \in L(p) \Leftrightarrow s \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi, \sigma)$.

$\varphi = x$: Inspecting the second case of the algorithm we get $\mathcal{K}, s, \sigma \models \varphi \Leftrightarrow s = \sigma(x) \Leftrightarrow s \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi, \sigma)$.

Now, assume that the statement already holds for φ', φ'' and assume that $\varphi = \neg\varphi'$. Then we get $\mathcal{K}, s, \sigma \models \varphi \Leftrightarrow \mathcal{K}, s, \sigma \not\models \varphi'$ by the semantics of $\mathrm{HCTL}^*_{\mathsf{ss}}$ and we have $\mathcal{K}, s, \sigma \not\models \varphi' \Leftrightarrow s \notin \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma)$ by assumption. Finally, we have $s \notin \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma) \Leftrightarrow s \in S \setminus \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi', \sigma) \Leftrightarrow s \in \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi, \sigma)$ by the negation clause in Algorithm 1.
The cases for $\varphi = \varphi' \vee \varphi''$, $\varphi = {\downarrow} x.\varphi'$ and $\varphi = @_x\, \varphi'$ can be shown in the same way. And lastly, assume that $\varphi = \mathsf{A}\psi$. Let $\varphi_1, \ldots, \varphi_m$ be the maximal state subformulas of ψ. Let $p_1, \ldots, p_m$ be fresh atomic propositions that do not occur in φ. Since $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$, the formula $\widetilde{\psi} := \psi[p_1/\varphi_1, \ldots, p_m/\varphi_m]$ is a pure LTL formula over $\mathit{Prop} \cup \{p_1, \ldots, p_m\} \cup \mathit{Var}$.
Let $\mathcal{K}'$ be the Kripke structure obtained by extending the labeling from $\mathcal{K}$ with $L(p_i) = \mathrm{MC}_{\mathsf{ss}}(\mathcal{K}, \varphi_i, \sigma)$ for $i \in \{1, \ldots, m\}$ and $L(x) = \{\sigma(x)\}$.
By a separate induction on $\widetilde{\psi}$ we show that for all paths $\pi \in \mathsf{Paths}(\mathcal{K})$ we have $\mathcal{K}, \pi \models \psi$ if and only if $\mathcal{K}', \pi \models \widetilde{\psi}$. The base cases for $\widetilde{\psi} = p, p_i, x$ follow by construction of $\mathcal{K}'$ and the induction hypothesis for φ_i. Negation, disjunction and temporal operators then follow by simple semantical considerations.
Notice that in the case of $\mathsf{A}\psi$ the labeling of the Kripke structure gets updated first with the values for the fresh atomic propositions and second with the values for the variables as done here. Because of that the LTL model checker does not need to distinguish between variables and atomic propositions. Correctness of the algorithm for the case $\mathsf{A}\psi$ then follows immediately. This finishes the proof. $\square$

Theorem 6.20. The model checking problem for $\mathrm{HCTL}^*_{\mathsf{ss}}$ is in PSPACE.

Proof. The space needed by all operations except for the case of $\mathsf{A}\psi$ is at most linear in $|\mathcal{K}| + |\varphi|$. The space needed for the case $\mathsf{A}\psi$ is dominated by the LTL model checker which can be done in space polynomial in $|\mathcal{K}| + |\varphi|$. Lastly, the inputs can be stored in space that is linear in $|\mathcal{K}| + |\varphi|$. Thus, the space needed to store all inputs in the recursion is also bounded polynomially in $|\mathcal{K}| + |\varphi|$. Thus, Algorithm 1 can be implemented to run in PSPACE. □

In summary we get the following results for $\mathrm{HCTL}^*_{\mathsf{ss}}$ and its fragments for model checking.

Corollary 6.21. Model Checking for HCTL, HCTL^+, HFCTL^+ and $\mathrm{HCTL}^*_{\mathsf{ss}}$ is PSPACE-complete, even for expression complexity.

On the other hand, when formulas are fixed, model checking becomes significantly easier. In fact, it is NLOGSPACE-complete. However, with the template above in Algorithm 1 this is not easily achievable. In the $\mathsf{A}\psi$ case we relabel possibly the whole structure with new atomic propositions for the maximal state subformulas in a path formula. The space needed for this relabelling may be up to linear in the size of the structure and thus Algorithm 1 is in PSPACE even for fixed formulas (although its runtime the is still polynomial). Such a bottom-up approach is not optimal for the data-complexity. The same issue was also observed for the data complexity of CTL^* for which an NLOGSPACE procedure was achieved in [60] via an automata-theoretic approach that can also be seen as a top-down approach.
We will present a top-down approach that also leads to an NLOGSPACE-procedure for $\mathrm{HCTL}^*_{\mathsf{ss}}$ at the end of Section 6.2.2. For this we will restrict the model checking games for $\mathrm{HCTL}^*_{\mathsf{ps}}$ and the associated techniques for solving them which will be introduced in the next section. We still state the result here but prove it later.

Corollary 6.22. The data complexity of HCTL, HCTL^+, HFCTL^+ and $\mathrm{HCTL}^*_{\mathsf{ss}}$ is NLOGSPACE-complete.

6.2.2 $\mathrm{HCTL}^*_{\mathsf{ps}}$

As we have seen in Subsection 6.1.2, model checking for $\mathrm{HCTL}^*_{\mathsf{ps}}$ is significantly harder than that of $\mathrm{HCTL}^*_{\mathsf{ss}}$. The model checking problem is EXPSPACE-hard and even the data complexity rose to at least PSPACE.
Moreover, the recursive algorithm for $\mathrm{HCTL}^*_{\mathsf{ss}}$ is not a convenient template that can easily be expanded to $\mathrm{HCTL}^*_{\mathsf{ps}}$. Remember that $\mathrm{HCTL}^*_{\mathsf{ps}}$ allows binders to range over arbitrary path formulas. As a consequence, to employ this template we would basically need a model checking algorithm for LTL

with added variables and binders interpreted over paths of arbitrary Kripke structures.
However, this particular semantics for hybrid LTL seems to be an open problem. This does not mean that hybrid linear temporal logic and its model checking problem has not yet been studied. For example the authors of [83] prove PSPACE-completeness for the model checking problem of an LTL-like logic, but only over linear structures, i.e. infinite words. Hence, their interpretation of binders significantly deviates from ours in that a variable is only bound to a single moment along the path and not to a state in the structure which might occur more than once along the path. In [37] the model checking problem of a logic with LTL syntax is also investigated and PSPACE-completeness for this problem is proven and [62] later improved upon their techniques by adding a game-based model checking algorithm. However despite the LTL-like syntax, the semantics of the logic in both papers is more CTL-like and thus also does not fit our problem.
So instead of building upon this recursive algorithm we characterise the model checking problem of $\mathrm{HCTL}^*_{\mathsf{ps}}$ by a game-based framework which builds upon CTL^* model checking games (cf. [65, 11]) and show how to solve these games in EXPSPACE. The soundness and completeness proofs as well as the algorithms used to solve these model checking games adapt techniques presented in [24] which present similar model checking games for CTL^*.

Model Checking Games. Let us fix a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$, a state $s \in S$ and a variable assignment $\sigma : \mathit{Var} \to S$ and a formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ps}}$ in negation normal form for the remainder of this section. Since we then do not have full negation on path formulas and to still have a branching-time logic that is temporally complete, we instead treat the Release-operator R – usually defined through $\psi_1 \mathsf{R} \psi_2 := \neg(\neg\psi_1 \mathsf{U} \neg\psi_2)$ – as a native operator.

Definition 6.23. The model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ is played between player $\boldsymbol{V}$ and $\boldsymbol{R}$. Intuitively, $\boldsymbol{V}$ tries to prove that $\mathcal{K}, s, \sigma \models \varphi$ holds while $\boldsymbol{R}$ tries to refute this.
The game is played on the configuration space $S \times \{\mathsf{E}, \mathsf{A}\} \times 2^{\mathsf{Fl}(\varphi) \times (\mathit{Var} \to S)}$ and we usually denote a configuration in the form $s \vdash Q([\varphi_1]^{\sigma_1}, \ldots, [\varphi_m]^{\sigma_m})$ where $s \in S$, $Q \in \{\mathsf{E}, \mathsf{A}\}$, $\sigma_i : \mathit{Var} \to S$ and $\varphi_i \in \mathsf{Fl}(\varphi)$ for $i \in \{1, \ldots, m\}$.
A play is a sequence of configurations starting with the configuration $s \vdash \mathsf{A}([\varphi]^{\sigma})$ and evolving according to the rules depicted in Figure 6.5. The rules are to be read top-down, i.e. if a play reaches a configuration that matches the pattern in the upper part of the rule, this induces a choice for the player annotated on the side of the rule (similar to the model checking games for H_μ introduced in Section 3.5.3). The choice involves either picking one of the

successor configurations depicted in the lower part of the rule or choosing a successor of the state in the current configuration.
Rules with no player annotated at the side are deterministic and can be associated with either player.

Note that at any given time in the game more than one rule might be possible to apply since the rules only need the occurrence of certain formula types. However, the order in which these rules might be carried out does not affect the winner of the game as will get clearer when discussing the winning conditions and soundness and completeness of these games.
The winning condition on these plays is split between finite and infinite plays. The winning condition for finite plays is determined through its final configuration. However, the winning conditions for infinite plays are more involved and deal with the unfolding of U and R formulas, since they are the only rule applications that increase the number of subformulas in a configuration and thus they are the only ones that can create an infinite play. Intuitively, in an infinite play $\boldsymbol{V}$ wants to make sure that U-formulas are satisfied after a finite amount of time while R-formulas can safely be unfolded infinitely often. However, the type of the configuration also plays a role, e.g. whether *some* infinitely regenerating R-formula is enough to show the satisfaction of a formula or if *all* infinitely regenerating formulas need to be R-formulas.
The following definitions and lemmas make this intuition precise.

Definition 6.24. Let $(s_i \vdash Q_i\Gamma_i)_{i\in\mathbb{N}}$ be an infinite play. It is called an E-, resp. A-play if there is an $n \in \mathbb{N}$ such that $Q_i = \mathsf{E}$ resp. $Q_i = \mathsf{A}$ for all $i \geq n$.

Lemma 6.25. Every infinite play is either an E-play or an A-play.

Proof. Let $(s_i \vdash Q_i\Gamma_i)_{i\in\mathbb{N}}$ be an infinite play and let $(n_i)_{i\in\mathbb{N}}$ be the sequence of the sums of all subformulas occurring in Γ_i in each step of the play, i.e. $n_i \coloneqq \sum_{[\varphi]^\sigma\in\Gamma_i} |\varphi|$.
First, for the game to be infinite it must be the case that $n_i > 0$ for all $i \in \mathbb{N}$. Otherwise the game would be stuck with no rule applicable. Now, observe that all rules except for the U- and R-rules strictly decrease n_i if they are applied to $s_i \vdash Q_i\Gamma_i$. Thus, for the play to be infinite there are infinitely many of the U- and R-rule applications and they are also the only rule applications that possibly *add* subformulas to the next configuration.
Secondly, observe that the rules (AQ) and (EQ) strictly decrease the number of path quantifiers occurring in some configuration and the U- and R-rules (the only rules that *add* subformulas to a play) do not generate additional path-quantifiers. Thus, the rules (AQ) and (EQ) can occur only finitely often in an infinite play. However, this means that after the last of finitely many

$$\frac{s \vdash \mathsf{E}([\varphi_1 \vee \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi_1]^{\sigma}, \Gamma) \quad s \vdash \mathsf{E}([\varphi_2]^{\sigma}, \Gamma)}\ \boldsymbol{V} \qquad \frac{s \vdash \mathsf{A}([\varphi_1 \vee \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi_1]^{\sigma}, [\varphi_2]^{\sigma}, \Gamma)}$$

$$\frac{s \vdash \mathsf{E}([\varphi_1 \wedge \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi_1]^{\sigma}, [\varphi_2]^{\sigma}, \Gamma)} \qquad \frac{s \vdash \mathsf{A}([\varphi_1 \wedge \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi_1]^{\sigma}, \Gamma) \quad s \vdash \mathsf{A}([\varphi_2]^{\sigma}, \Gamma)}\ \boldsymbol{R}$$

$$\frac{s \vdash \mathsf{E}([p]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}(\Gamma)}\ \text{if } s \in L(p) \qquad \frac{s \vdash \mathsf{A}([p]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}(\Gamma)}\ \text{if } s \notin L(p)$$

$$\frac{s \vdash \mathsf{E}([x]^{\sigma}, \Gamma)}{s \vdash_{\sigma} \mathsf{E}(\Gamma)}\ \text{if } \sigma(x) = s \qquad \frac{s \vdash \mathsf{A}([x]^{\sigma}, \Gamma)}{s \vdash_{\sigma} \mathsf{A}\Gamma}\ \text{if } \sigma(x) \neq s$$

$$(\mathsf{EQ})\ \frac{s \vdash \mathsf{E}([Q\psi]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}(\Gamma) \quad s \vdash Q([\psi]^{\sigma})}\ \boldsymbol{R} \qquad (\mathsf{AQ})\ \frac{s \vdash \mathsf{A}([Q\psi]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}(\Gamma) \quad s \vdash Q([\psi]^{\sigma})}\ \boldsymbol{V}$$

$$\frac{s \vdash \mathsf{E}([\downarrow x.\varphi]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi]^{\sigma[x \to s]}, \Gamma)} \qquad \frac{s \vdash \mathsf{A}([\downarrow x.\varphi]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi]^{\sigma[x \to s]}, \Gamma)}$$

$$\frac{s \vdash \mathsf{E}([@_x\varphi]^{\sigma}, \Gamma)}{\sigma(x) \vdash \mathsf{E}([\varphi]^{\sigma}) \quad s \vdash \mathsf{E}(\Gamma)}\ \boldsymbol{R} \qquad \frac{s \vdash \mathsf{A}([@_x\varphi]^{\sigma}, \Gamma)}{\sigma(x) \vdash \mathsf{A}([\varphi]^{\sigma}) \quad s \vdash \mathsf{A}(\Gamma)}\ \boldsymbol{V}$$

$$(\mathsf{EX})\ \frac{s \vdash \mathsf{E}([\mathsf{X}\varphi_1]^{\sigma_1}, \ldots, [\mathsf{X}\varphi_k]^{\sigma_k})}{t \vdash \mathsf{E}([\varphi_1]^{\sigma_1}, \ldots, [\varphi_k]^{\sigma})}\ \boldsymbol{V} : s \to t$$

$$(\mathsf{AX})\ \frac{s \vdash \mathsf{A}([\mathsf{X}\varphi_1]^{\sigma_1}, \ldots, [\mathsf{X}\varphi_k]^{\sigma_k})}{t \vdash \mathsf{A}([\varphi_1]^{\sigma_k}, \ldots, [\varphi_k]^{\sigma_k})}\ \boldsymbol{R} : s \to t$$

$$(\mathsf{EU})\ \frac{s \vdash \mathsf{E}([\varphi_1 \mathsf{U} \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi_2]^{\sigma}, \Gamma) \quad s \vdash \mathsf{E}([\varphi_1]^{\sigma}, [\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)]^{\sigma}, \Gamma)}\ \boldsymbol{V}$$

$$\frac{s \vdash \mathsf{E}([\varphi_1 \mathsf{R} \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi_2]^{\sigma}, [\varphi_1]^{\sigma}, \Gamma) \quad s \vdash \mathsf{E}([\varphi_2]^{\sigma}, [\mathsf{X}(\varphi_1 \mathsf{R} \varphi_2)]^{\sigma}, \Gamma)}\ \boldsymbol{V}$$

$$\frac{s \vdash \mathsf{A}([\varphi_1 \mathsf{U} \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi_2]^{\sigma}, [\varphi_1]^{\sigma}, \Gamma) \quad s \vdash \mathsf{A}([\varphi_2]^{\sigma}, [\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)]^{\sigma}, \Gamma)}\ \boldsymbol{R}$$

$$\frac{s \vdash \mathsf{A}([\varphi_1 \mathsf{R} \varphi_2]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi_2]^{\sigma}, \Gamma) \quad s \vdash \mathsf{A}([\varphi_1]^{\sigma}, [\mathsf{X}(\varphi_1 \mathsf{R} \varphi_2)]^{\sigma}, \Gamma)}\ \boldsymbol{R}$$

Figure 6.5: The model checking game rules for HCTL$^*_{\mathsf{ps}}$.

rule applications of (AQ) or (EQ) the path-quantifier stays the same for the rest of the play, i.e. making it either an A- or E-play. $\square$

Definition 6.26. Let $(s_i \vdash Q_i\Gamma_i)_{i\in\mathbb{N}}$ be an infinite play and let $(r_i)_{i\in\mathbb{N}}$ be the sequence of rule applications between those configurations. We call a sequence $([\varphi_i]^{\sigma_i})_{i\in\mathbb{N}}$ a *thread* if and only if

- $\varphi_i \in \Gamma_i$ for all $i \in \mathbb{N}$

and one of the following conditions holds for every pair $([\varphi_i]^{\sigma_i}, [\varphi_{i+1}]^{\sigma_{i+1}})$ that belongs to the sequence of configurations:

- the rule r_i replaces $[\varphi_i]^{\sigma_i}$ with $[\varphi_{i+1}]^{\sigma_{i+1}}$ (and possibly other formulas) inside the outer path quantifier,
- $\varphi_i = Q\varphi_{i+1}$ and r_i is (AQ) or (EQ) for $\mathsf{Q} \in \{\mathsf{E}, \mathsf{A}\}$,
- $\varphi_i = \mathsf{X}\varphi_{i+1}$ and r_i is (EX) or (AX) or
- $\varphi_i = \varphi_{i+1}$ and r_i operates on another formula.

We call a thread $([\varphi_i]^{\sigma_i})_{i\in\mathbb{N}}$ a μ-, resp. ν-thread if there is a formula $\varphi = \psi_1\mathsf{U}\psi_2$, resp. $\varphi = \psi_1\mathsf{R}\psi_2$ such that $\varphi = \varphi_i$ for infinitely many $i \in \mathbb{N}$.

Lemma 6.27. Every infinite play has at least one thread and every thread is either a μ-thread or a ν-thread.

Proof. Let $(s_i \vdash Q_i\Gamma_i)_{i\in\mathbb{N}}$ be an infinite play and let R be the binary relation described in Definition 6.26 that links two formulas in successive configurations. Then R forms a directed acyclic graph on top of the underlying play structure. Since every configuration consists of only finitely many formulas and variable assignments this DAG is finitely branching. Further we have paths of arbitrary length in this DAG because every formula in every configuration can be traced back via R to a single formula in the initial configuration of the play. Hence, by König's Lemma there is also an infinite path in this DAG, i.e. a thread.
To further prove that every thread is also a μ- or ν-thread, we need to strengthen an observation made in the proof of Lemma 6.25. We stated that every infinite play needs infinitely many unfoldings of U- or R-formulas via the respective game rules. Particularly, this means that there are infinitely many applications of these rules that apply the rightmost choice, producing a $\mathsf{X}(\psi_1\mathsf{U}\psi_2)$ resp. $\mathsf{X}(\psi_1\mathsf{R}\psi_2)$ formula. During the infinite play, these added subformulas need to be resolved at some point (since there are only finitely many that can be unfolded) using the rules (EX) or (AX). Thus, every infinite

play has also infinitely many applications of the rules (EX) or (AX). However, when using (EX) or (AX), *all* formulas have the topmost connective X. Hence, every infinite thread must contain infinitely many formulas of the type $\mathsf{X}\psi$. Since (EX) or (AX) strictly decrease the size of the formula on a thread and again as in Lemma 6.25 the unfolding rules for U and R are the only rules that may increase the size of a formula on a thread, there also have to be infinitely many of these rule applications along a thread. Hence, each thread is at least a μ- or ν-thread.
To see that a thread cannot be a μ- and a ν-thread, observe that to change from a U-formula to a R-formula (or vice versa) along a thread one has to be a strict subformula of the other, thus such a change strictly decreases the size of the formulas along the thread and thus can only occur finitely often. $\square$

With these observations we can give the winning conditions for the model checking games.

Definition 6.28. Let $(C_i)_{i\in I}$ with $I = \{1, \ldots, n\}$ or $I = \mathbb{N}$ be a finite or infinite play of $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$. Then $\boldsymbol{V}$ wins a finite play, if

- $C_n = s \vdash \mathsf{E}\emptyset$ for some $n \geq 0$,
- $C_n = s \vdash \mathsf{A}([p]^\rho, \Gamma)$ with $s \in L(p)$ for some $n \geq 0$ or
- $C_n = s \vdash \mathsf{A}([x]^\rho, \Gamma)$ with $\rho(x) = s$ for some $n \geq 0$.

And $\boldsymbol{V}$ wins an infinite play, if

- the play is an E-play and contains no μ-thread or
- the play is an A-play and contains a ν-thread.

On the other hand, $\boldsymbol{R}$ wins in the other cases, namely: $\boldsymbol{R}$ wins a finite play, if

- $C_n = s \vdash \mathsf{A}\emptyset$ for some $n \geq 0$,
- $C_n = s \vdash \mathsf{E}([p]^\rho, \Gamma)$ with $s \notin L(p)$ for some $n \geq 0$ or
- $C_n = s \vdash \mathsf{E}([x]^\rho, \Gamma)$ with $\rho(x) \neq s$ for some $n \geq 0$.

And finally $\boldsymbol{R}$ wins an infinite play, if

- the play is an E-play and contains a μ-thread or
- the play is an A-play and contains no ν-thread.

Using Lemmas 6.25 and 6.27 it is easy to see that the winning conditions are well-defined and each play has a unique winner.

Example 6.29. Consider the formula $\varphi := \mathsf{E}(\downarrow x.\mathsf{XG}\neg x \wedge \mathsf{X}\downarrow x.\mathsf{F}x)$ and the structure

$$\begin{tikzcd} \mathcal{K}: & {} \arrow[r] & s_0 \arrow[r] & s_1 \arrow[loop right] \end{tikzcd}$$

The arena of the model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s_0, \sigma, \varphi)$ is depicted in Figure 6.6. To ease notation we have marked the configurations in which player $\boldsymbol{V}$ has a choice to either unfold a G- or F-operator. Since there is no overlap with $\boldsymbol{R}$'s choices this is possible in this particular game instance. Furthermore, we have neglected some unfoldings of the G-operator that would have ended in a configuration that is immediately lost by $\boldsymbol{V}$ like the one marked by a dashed arrow in the top left of Figure 6.6. Since all choices in this game are made by $\boldsymbol{V}$ we can be sure that these will never be taken anyways.
We can see that there are no finite plays that can be won by $\boldsymbol{V}$ and there are only two possibilities for infinite plays. Either a play can end up in the bottom three states $s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0})$, $s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0})$ or $s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{XG}\neg x]^{x\mapsto s_0})$ in which it then gets trapped, or $\boldsymbol{V}$ can decide to never unfold $[\mathsf{F}x]^{x\mapsto s_1}$ to $[x]^{x\mapsto s_1}$. In this case the game cycles indefinitely somewhere in the states depicted in the middle of Figure 6.6. In particular the configuration $s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0}, [\mathsf{F}x]^{x\mapsto s_1})$ is seen infinitely often.
In both cases the infinite play is an E-play. The first case is won by player $\boldsymbol{V}$ since the only regenerating formula is a G-formula and thus forming a ν-thread. Since there is no infinitely regenerating U-formula there is no μ-thread in this play. The second case however is won by player $\boldsymbol{R}$ since then the infinite unfolding of the block $[\mathsf{F}x]^{x\mapsto s_1}$ forms a μ-thread.
Since $\boldsymbol{R}$ has no choices in this particular game it is obvious that Duplicator has a winning strategy. A close inspection of the structure and the formula also reveals that $\mathcal{K}, s_0 \models \varphi$ independently of the initial variable assignment.

Soundness and Completeness. The following paragraphs show soundness and completeness of the model checking games for $\mathrm{HCTL}^*_{\mathsf{ps}}$.

Definition 6.30. Let $C := s \vdash \mathsf{E}(\Gamma)$ be a configuration in a model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$. We call C *true* if there is a path π with $\pi^0 = s$ such that $\mathcal{K}, \pi, \sigma \models \chi$ for every $[\chi]^\sigma \in \Gamma$. Analogously we call a configuration $s \vdash \mathsf{A}(\Gamma)$ *true* if for all paths π with $\pi^0 = s$ there is a $[\chi]^\sigma \in \Gamma$ such that $\mathcal{K}, \pi, \sigma \models \chi$. A configuration is called *false* if it is not true.

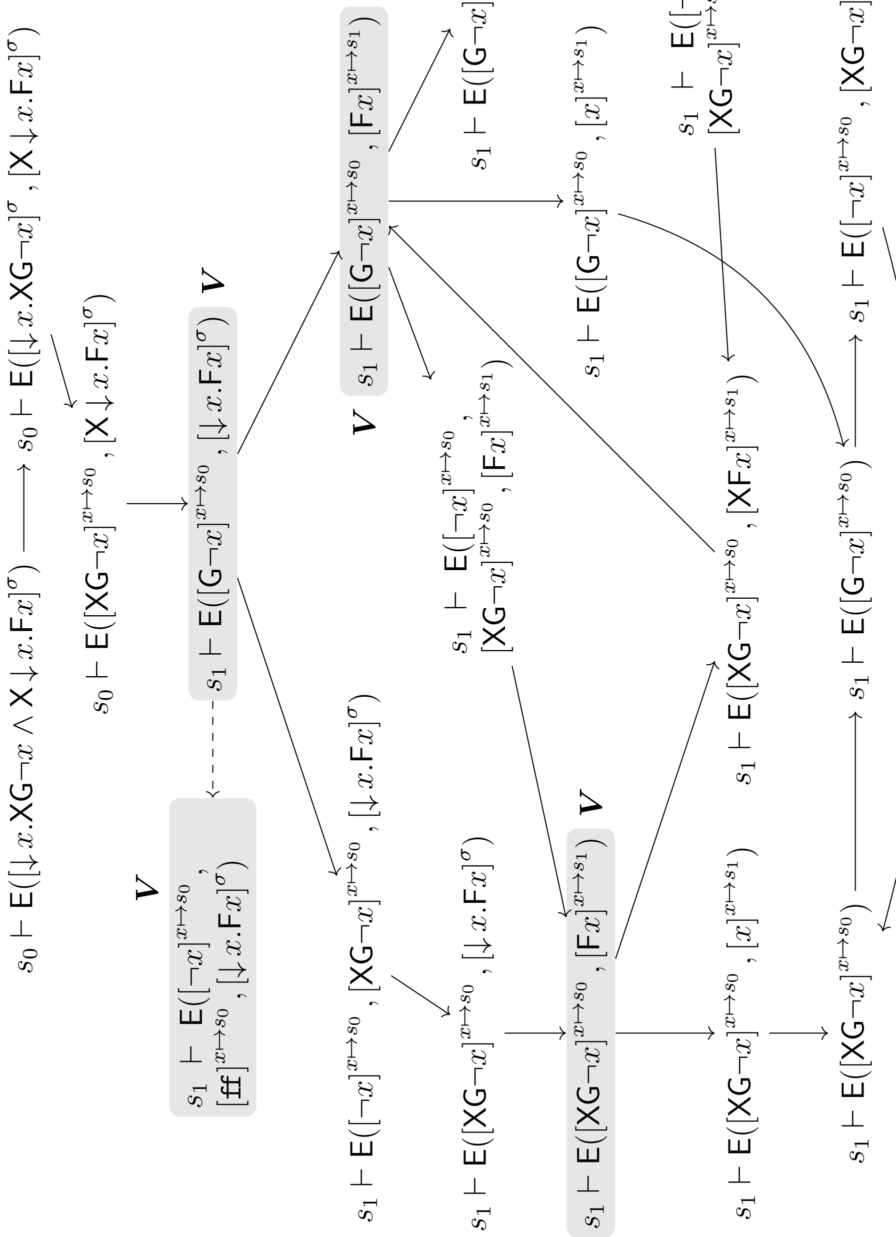

$s_0 \vdash \mathsf{E}([\downarrow x.\mathsf{XG}\neg x \wedge \mathsf{X}\downarrow x.\mathsf{F}x]^\sigma)$
$s_0 \vdash \mathsf{E}([\downarrow x.\mathsf{XG}\neg x]^\sigma, [\mathsf{X}\downarrow x.\mathsf{F}x]^\sigma)$
$s_0 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0}, [\mathsf{X}\downarrow x.\mathsf{F}x]^\sigma)$
$s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0}, [\downarrow x.\mathsf{F}x]^\sigma)$
V
$s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{ff}]^{x\mapsto s_0}, [\downarrow x.\mathsf{F}x]^\sigma)$
V
$s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{XG}\neg x]^{x\mapsto s_0}, [\downarrow x.\mathsf{F}x]^\sigma)$
$s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0}, [\downarrow x.\mathsf{F}x]^\sigma)$
$s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0}, [\mathsf{F}x]^{x\mapsto s_1})$
V
$s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0}, [x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0})$
$s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0}, [\mathsf{F}x]^{x\mapsto s_1})$
V
$s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{XG}\neg x]^{x\mapsto s_0}, [\mathsf{F}x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\mathsf{XG}\neg x]^{x\mapsto s_0}, [\mathsf{XF}x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0}, [\mathsf{XF}x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0}, [x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{XG}\neg x]^{x\mapsto s_0}, [\mathsf{XF}x]^{x\mapsto s_1})$
$s_1 \vdash \mathsf{E}([\mathsf{G}\neg x]^{x\mapsto s_0})$
$s_1 \vdash \mathsf{E}([\neg x]^{x\mapsto s_0}, [\mathsf{XG}\neg x]^{x\mapsto s_0})$

Figure 6.6: The model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s_0, \sigma, \mathsf{E}(\downarrow x.\mathsf{XG}\neg x \wedge \mathsf{X}\downarrow x.\mathsf{F}x))$

Lemma 6.31. Let C be a configuration in a model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$. If C is true then it holds that:

- All deterministic rules yield a successor configuration that is true.
- All rules in which $\boldsymbol{V}$ has to choose have a successor configuration that is true.
- All successor configurations of rules that belong to $\boldsymbol{R}$ are true.

Analogously, if C is false then

- All deterministic rules yield a successor configuration that is false.
- All rules in which $\boldsymbol{R}$ has to choose have a successor configuration that is false.
- All successor configurations of rules that belong to $\boldsymbol{V}$ are false.

Proof. The proof is a simple case distinction over all possible rules. We will only showcase a few examples preserving truth.
Suppose that $C = s \vdash \mathsf{E}([\varphi_1 \wedge \varphi_2]^\sigma, \Gamma)$ is a true configuration. Then there is a path π with $\pi^0 = s$ such that $\mathcal{K}, \pi, \sigma \models \varphi_1 \wedge \varphi_2$. By the semantics of $\text{HCTL}^*_{\mathsf{ps}}$ we get that $\mathcal{K}, \pi, \sigma \models \varphi_1$ and $\mathcal{K}, \pi, \sigma \models \varphi_2$. Since satisfaction of formulas in Γ stays the same we get that $s \vdash \mathsf{E}([\varphi_1]^\sigma, [\varphi_2]^\sigma, \Gamma)$ is a true configuration as well.
Suppose now that $C = s \vdash \mathsf{E}([\varphi_1 \mathsf{U} \varphi_2]^\sigma, \Gamma)$ is a true configuration. Thus, there is a path π with $\pi^0 = s$ such that $\mathcal{K}, \pi, \sigma \models \varphi_1 \mathsf{U} \varphi_2$. We also know that $\varphi_1 \mathsf{U} \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2))$. The latter disjunction is $\boldsymbol{V}$'s choice in the game rule (EU). Thus, depending on π either the first or second successor configuration is true.
Lastly, suppose that $C = s \vdash \mathsf{E}([@_x\, \varphi_1]^\sigma, \Gamma)$ is a true configuration. Thus, there is a path π with $\pi^0 = s$ such that $\mathcal{K}, \pi, \sigma \models @_x\, \varphi_1$ and $\mathcal{K}, \pi, \sigma' \models \varphi'$ for every $[\varphi']^{\sigma'} \in \Gamma$. By the semantics of $\text{HCTL}^*_{\mathsf{ps}}$ and since $@_x\, \varphi_1$ is a state formula we have that $\mathcal{K}, \sigma(x), \sigma \models \varphi_1$ and thus $\sigma(x) \vdash \mathsf{E}([\varphi_1]^\sigma)$ is a true configuration. Also $s \vdash \mathsf{E}(\Gamma)$ is also still a true configuration witnessed by π.
The remaining cases for true configurations and dual cases for false configurations can be shown in a similar fashion. $\square$

This lemma forms the basis of a winning strategy for either player in proving soundness and completeness of the model checking games for $\text{HCTL}^*_{\mathsf{ps}}$.

Theorem 6.32. Player $\boldsymbol{V}$ has a winning strategy in $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ if and only if $\mathcal{K}, s, \sigma \models \varphi$.

Proof. We will only show the if-direction. The only if-direction is almost identical because of the duality of true and false configurations and game rules.
So, suppose that $\mathcal{K}, s, \sigma \models \varphi$. Then, since φ is a state formula we can deduce that $s \vdash \mathsf{A}([\varphi]^\sigma)$ is a true configuration. We now describe a winning strategy for $\boldsymbol{V}$:

- On false configurations $\boldsymbol{V}$ can simply make an arbitrary choice.
- On true configurations $\boldsymbol{V}$ makes a choice that preserves truth according to Lemma 6.31. If this choice is done with the rule (EU) and both possible choices preserve truth, then $\boldsymbol{V}$ chooses the first option over the second, i.e. $\boldsymbol{V}$ tries to satisfy each Until-formula as soon as possible by unfolding $\varphi_1 \mathsf{U} \varphi_2$ preferably to φ_2.

Using Lemma 6.31 we deduce that the play starting at $s \vdash \mathsf{A}([\varphi]^\sigma)$ which is played according to this strategy only visits true configurations. To determine the winner of this play, notice that for finite games, $\boldsymbol{R}$ only wins at false configurations. Thus, every finite play according to this strategy is won by $\boldsymbol{V}$.
So, suppose that $\lambda := (s_i \vdash Q_i\Gamma_i)_{i\in\mathbb{N}}$ is an infinite play in which $\boldsymbol{V}$ plays according to the strategy above. There are two cases depending on whether the play is an E-play or an A-play.
Suppose first that λ is an E-play and let $C_n = s_n \vdash \mathsf{E}(\Gamma_n)$ be the first configuration such that the defining path quantifier in the configuration does not change anymore. With the arguments above, C_n is a true configuration. Thus, there is a path π starting at s_n such that $\mathcal{K}, s_n, \sigma \models \varphi_1 \mathsf{U} \varphi_2$ for all $[\varphi_1 \mathsf{U} \varphi_2]^\sigma \in \Gamma_n$. Since the game is played according to the strategy above, $\boldsymbol{V}$ follows π with every (EX) rule application (there are infinitely many (EX) rule applications as discussed in Lemma 6.27). Note further that before every (EX) rule application *all* other possible rule applications need to be performed since (EX) needs all formulas to be underneath an X operator. Now, let $s_{n+k} \vdash \mathsf{E}(\Gamma_{n+k})$ be a configuration such that $\mathcal{K}, \pi^{[k,\infty)}, \sigma \models \varphi_2$ which exists since $s_n \vdash \mathsf{E}(\Gamma_n)$ was a true configuration. Then, before leaving this configuration with (EX) by the strategy above, $\boldsymbol{V}$ chooses the first option when applying the rule (EU) to $\varphi_1 \mathsf{U} \varphi_2$. In all following configurations $\varphi_1 \mathsf{U} \varphi_2$ will not appear again. Hence, all Until-formulas in the E-play disappear after a finite number of steps. Thus, no thread in the play can be a μ-thread and $\boldsymbol{V}$ wins.
Lastly, suppose that λ is an A-play and again let n be the moment such that the defining path quantifier in the configurations starting with C_n does not change anymore. As stated above C_n must be a true configuration. Let

$\pi = s_n, s_{n+1}, s_{n+2} \dots$ be the path that $\boldsymbol{R}$ chooses to play with the rules (AX) in this play. Since C_n is true there is some $[\chi]^\sigma \in \Gamma_n$ such that $\mathcal{K}, \pi, \sigma \models \chi$. Suppose, $\chi = \varphi_1 \mathsf{U} \varphi_2$. It is easy to see that then there is a configuration C_{n+j} for some $j \in \mathbb{N}$ such that $[\varphi_2]^\sigma \in \Gamma_{n+j}$ and $\mathcal{K}, \pi^{[j,\infty)}, \sigma \models \varphi_2$ or $[\varphi_1]^\sigma \in \Gamma_{n+j}$ and $\mathcal{K}, \pi^{[j,\infty)}, \sigma \models \varphi_1$. This is because before each (AX) rule application $\boldsymbol{R}$ has to unfold $[\varphi_1 \mathsf{U} \varphi_2]^\sigma$ with rule (AU). Since $\mathcal{K}, \pi, \sigma \models \varphi_1 \mathsf{U} \varphi_2$, either φ_1 or φ_2 holds here, so the statement follows if $\boldsymbol{R}$ picks the left alternative. If $\boldsymbol{R}$ picks the right alternative then he produces $[\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)]^\sigma$ which gets unfolded to $[\varphi_1 \mathsf{U} \varphi_2]^\sigma$ after the next (AX) rule. This repeats until at some point φ_2 is satisfied along π in which case both alternatives of rule (AU) produce $[\varphi_2]^\sigma$. Now the key point is that both φ_1 and φ_2 are strictly smaller than $\varphi_1 \mathsf{U} \varphi_2$. Now φ_1 or φ_2 can be unfolded again depending on its top connective and so on. Thus, if χ only contains Until-formulas that successively get unfolded then at some point in time we end up in a true configuration C_m where $m \geq n$ with a propositional literal which contradicts the assumption that we have an infinite A-play.
Thus, there needs to be some R-formula that gets unfolded via the second alternative in rule (AR) infinitely often (the first choice in rule (AR) also leads to strictly smaller formulas as before) since this is the only remaining option in the game rules that increases the size of the formulas in the configuration (as opposed to infinitely often reducing the size of the formulas via the rule (AX)). It is not hard to see that following this R-formula creates a ν-thread and thus $\boldsymbol{V}$ wins. □

Solving Model Checking Games. Having a game-based framework for model checking does not immediately yield an algorithm to solve these games. We will now show how to reduce these games to parity games. We can then simply employ one of the multitude of algorithms solving parity games, cf. [49, 94, 82, 51, 39, 20], which will yield an ExpSpace model checking algorithm that matches our lower bound. The reduction is achieved by showing that the winning condition for these model checking games can be equivalently formulated as an ω-regular winning condition, i.e. a winning condition described by a parity automaton. This then immediately yields the reduction to parity games via a simple product construction of the game and the automaton for the winning condition as shown in Proposition 2.48.
We fix a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L \rangle$ and a formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ps}}$ for the remainder of this section and begin by defining a symbolic alphabet of rule applications that is designed to model the rule applications throughout a play in the model checking game as described in Section 2.9.2.

Definition 6.33. Let $\varphi \in \mathrm{HCTL}^*_{\mathsf{ps}}$. Then we define the alphabet of symbolic rule applications in the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ as

$$\begin{aligned}
\Sigma_{\varphi,\mathcal{K}} := \; & \{\mathsf{EOR}_d(\varphi_1, \varphi_2, \sigma), \mathsf{AOR}(\varphi_1, \varphi_2, \sigma), \\
& \quad \mathsf{EAND}(\varphi_1, \varphi_2, \sigma), \mathsf{AAND}_d(\varphi_1, \varphi_2, \sigma), \\
& \quad \mathsf{EU}_d(\varphi_1, \varphi_2, \sigma), \mathsf{AU}_d(\varphi_1, \varphi_2, \sigma), \\
& \quad \mathsf{ER}_d(\varphi_1, \varphi_2, \sigma), \mathsf{AR}_d(\varphi_1, \varphi_2, \sigma) \mid \\
& \qquad \varphi_1, \varphi_2 \in \mathsf{Fl}(\varphi), \sigma : \mathit{Var} \to S, d \in \{\mathsf{lft}, \mathsf{rgh}\}\} \\
& \cup \{\mathsf{EQ}_d([\mathsf{Q}\psi]^\sigma), \mathsf{AQ}_d([\mathsf{Q}\psi]^\sigma) \mid \mathsf{Q} \in \{\mathsf{E}, \mathsf{A}\}, \mathsf{Q}\psi \in \mathsf{Fl}(\varphi), \\
& \qquad \sigma : \mathit{Var} \to S, d \in \{\mathsf{lft}, \mathsf{rgh}\}\} \\
& \cup \{\mathsf{EX}, \mathsf{AX}\} \\
& \cup \{\mathsf{ELit}([l]^\sigma), \mathsf{ALit}([l]^\sigma) \mid l \text{ Literal in } \mathsf{Fl}(\varphi), \sigma : \mathit{Var} \to S\} \\
& \cup \{\mathsf{EVar}([x]^\sigma), \mathsf{AVar}([x]^\sigma) \mid x \in \mathit{Var}, \sigma : \mathit{Var} \to S\} \\
& \cup \{\mathsf{EBind}([\varphi_1]^\sigma), \mathsf{ABind}([\varphi_1]^\sigma) \mid {\downarrow} x.\varphi_1 \in \mathsf{Fl}(\varphi), \\
& \qquad x \in \mathit{Var}, \sigma : \mathit{Var} \to S\} \\
& \cup \{\mathsf{EJump}([\varphi_1]^\sigma), \mathsf{AJump}([\varphi_1]^\sigma) \mid @_x \varphi_1 \in \mathsf{Fl}(\varphi), \\
& \qquad x \in \mathit{Var}, \sigma : \mathit{Var} \to S\} \\
& \cup \{\mathsf{Win}_{\boldsymbol{V}}, \mathsf{Win}_{\boldsymbol{R}}\}.
\end{aligned}$$

Note that $\Sigma_{\varphi,\mathcal{K}}$ is bounded by $\mathcal{O}((|\varphi| \cdot |\mathcal{K}|^{|\varphi|})^2)$, thus the alphabet is exponential in the size of φ.

Each alphabet symbol is naturally associated with a rule application in the model checking game. For instance the symbol $\mathsf{EU}_{lft}(\varphi_1, \varphi_2, \sigma)$ is associated with player $\boldsymbol{V}$'s choice to play the left option of rule (EU) on the principal formula $[\varphi_1 \mathsf{U} \varphi_2]^\sigma$. The symbols $\mathsf{Win}_{\mathsf{P}}$ are used to symbolically encode finite plays. They indicate that a final configuration was reached in which player P has won. For example the application of the rule

$$\frac{s \vdash \mathsf{A}([p]^\sigma, \Gamma)}{s \vdash \mathsf{E}(\Gamma)}$$

if $s \in L(p)$ is associated with $\mathsf{Win}_{\boldsymbol{V}}$.
Each play λ can be represented by the series of symbolic rule applications and thus as a word $w_\lambda \in \Sigma^\omega_{\varphi,\mathcal{K}}$ in which the i-th letter corresponds to the i-th rule application in the play. We assume that finite plays are encoded as words ending either with $(\mathsf{Win}_{\boldsymbol{V}})^\omega$ or $(\mathsf{Win}_{\boldsymbol{R}})^\omega$.
Note that we do not encode all information of a play. For instance we do not remember the state of a configuration in the symbolic representation of a play. Thus, we cannot uniquely identify a play by its associated word over

$\Sigma_{\varphi,\mathcal{K}}$. In turn, the automata for the winning condition will not be able to check if the encoded word truely symbolises a legal play. This however is not necessary since the automata for the winning conditions will only be used in a product construction with the game arena which already encodes the legal moves.

Next, we show that all winning plays for $\boldsymbol{V}$ can be accepted by a parity automaton. Finite plays are particularly easy since we use winning symbols for $\boldsymbol{V}$ and $\boldsymbol{R}$.

Theorem 6.34. There is a deterministic parity automaton $\mathcal{A}^{\mathsf{fin}}_{\varphi,\mathcal{K}}$ of constant size that accepts a finite play λ of the model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ if and only if λ is won by player $\boldsymbol{V}$.

Proof. By the above mentioned convention a symbolic representation of a finite play is won by player $\boldsymbol{V}$ iff it contains the symbold $\mathsf{Win}_{\boldsymbol{V}}$. This can easily be tested by a parity automaton with only two states. □

Note that we are using the fact that these automata are only supposed to work on plays. In general these automata accept even more words, especially some that may not even encode valid plays.

This first automaton only covers finite plays. For infinite plays we have to distinguish two types of plays, A-plays and E-plays. We will first deal with A-plays.

Theorem 6.35. There is a nondeterministic parity automaton $\mathcal{A}^{\mathsf{A}}_{\varphi,\mathcal{K}}$ of size $\mathcal{O}((|\varphi|\cdot|\mathcal{K}|^{|\varphi|})^2)$ that accepts an A-play λ of the model checking game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ if and only if λ is won by player $\boldsymbol{V}$.

Proof. Remember that $\boldsymbol{V}$ wins an A-play iff it contains a ν-thread. It is not hard to construct a nondeterministic parity automaton that simply guesses such a thread and follows it in its state space verifying that it is indeed a ν-thread in an A-play.

The automaton has an initial state that can read any finite prefix. It stays there until it guesses that it sees the last application of the rules (EQ) or (AQ), entering the infinite part of the A-play where the path-quantifier does not change anymore. The automaton then enters one component for each $\mathsf{A}\psi$ subformula and each σ depending on which of these subformulas was guessed. In each component the automaton then simply guesses a thread by following a particular subformula (together with its variable assignment) up to a point where it guesses that a $\chi_1 \mathsf{R} \chi_2$ subformula will be unfolded infinitely often. Then it enters a two-state component that simply verifies this.

The details of the construction are analogous to [24, Thm. 15.3.30] (with the added variable assignment that needs to be followed as well) and are omitted here.
The size of the automaton is dominated by the fact that it consists of at most $\mathcal{O}(|\varphi| \cdot |\mathcal{K}|^{|\varphi|})$ many components since $|\varphi| \cdot |\mathcal{K}|^{|\varphi|}$ is an upper bound on the number of $[\mathsf{A}\psi]^{\sigma}$-configurations that might be guessed by the automaton and each of these guessed configurations is followed by one component of the automaton that is again bounded by $|\varphi| \cdot |\mathcal{K}|^{|\varphi|}$. Also, already the alphabet $\Sigma_{\varphi,\mathcal{K}}$ used for this automaton has size $\mathcal{O}((|\varphi| \cdot |\mathcal{K}|^{|\varphi|})^2)$. Combining both yields the size estimation. □

The last automaton then deals with E-plays. The proof is again, analogous to [24, Thm. 15.3.31] and expands the concepts there by also dealing with added variable assignments.

Theorem 6.36. There is a deterministic parity automaton $\mathcal{A}^{\mathsf{E}}_{\varphi,\mathcal{K}}$ of size $\mathcal{O}(2^{|\varphi|\cdot|\mathcal{K}|^{|\varphi|}})$ that accepts an E-play λ of the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ if and only if λ is won by player $\boldsymbol{V}$.

Proof. The automaton uses as state set $\{\mathsf{E}, \mathsf{A}\} \times 2^{\mathsf{Fl}(\varphi)\times(Var\to S)} \times 2^{\mathcal{U}}$ where $\mathcal{U}$ describes the set of all U-subformulas in $\mathsf{Fl}(\varphi)$. The first two components of the state space essentially describe a configuration in the game and are solely used to track the game. For example in a configuration $(\mathsf{A}, [\psi \wedge \chi]^{\sigma}, \emptyset)$ and the next letter to be read is $\mathsf{AAND}_{rgh}([\psi]^{\sigma}, [\chi]^{\sigma})$ then the next state is $(\mathsf{A}, [\chi]^{\sigma}, \emptyset)$.
The last component of the state space is simply used to track the satisfaction of all Until formulas. The idea in tracking the satisfaction of Until-formulas is similar to the Miyano-Hayashi construction [71]. After seeing a rule that changes the path quantifier, the component is reset to $\emptyset$. If the rightmost component is empty, then after the next transition all Until-formulas which are present in the current configuration are added to the third component. Whenever the corresponding Until formula is satisfied through the application of the left part of the (EU)-rule then this specific Until-formula is deleted from the last component.
States in which the third component is $\emptyset$ receive priority 2 and all other states receive priority 1. It should be clear that an E-play is accepted if and only if each Until-formula is satisfied infinitely often, i.e. if the play does not contain a μ-thread.
The size estimation follows directly from the size of the state space of the automaton. □

We could now simply combine all three automata for the finite and infinite parts of the winning condition to obtain a nondeterministic parity automaton of doubly exponential size that accepts all the plays won by $\boldsymbol{V}$.
However, to use it in a product construction with the actual model checking game we would need another determinisation process magnifying its state space by another exponential. The product construction for games with ω-regular winning conditions described in Proposition 2.48 would then yield a parity game of size triply exponential for the model checking problem. Solving this however only yields a 3-EXPTIME upper bound which does not match the EXPSPACE lower bound proven in Theorem 6.9.
Instead we propose a more refined use of these automata to obtain an optimal decision procedure. First, we use a simple product construction for $\mathcal{A}^{\mathsf{fin}}_{\varphi,\mathcal{K}}$ and $\mathcal{A}^{\mathsf{A}}_{\varphi,\mathcal{K}}/\mathcal{A}^{\mathsf{E}}_{\varphi,\mathcal{K}}$ to obtain two automata for finite plays or E- resp. A-plays.

Corollary 6.37. For each $Q \in \{\mathsf{A}, \mathsf{E}\}$ there is a deterministic parity automaton (DPA) $\mathcal{A}^{Q}_{\mathcal{K},\varphi}$ of size doubly exponential in $|\varphi| + |\mathcal{K}|$ which, given an ω-sequence λ, accepts λ iff it is a finite play or an infinite Q-play that is won by player $\boldsymbol{V}$.

We then use both these automata and ideas from the decomposition method for CTL* model checking to obtain an optimal upper bound.

Theorem 6.38. The model checking problem for HCTL$^*_{\mathsf{ps}}$ is in EXPSPACE.

Proof. We describe a procedure for solving $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ that borrows ideas from the decomposition method for CTL* model checking sketched in Section 6.2.1 and uses at most exponential space.
Take the smallest subformulas of φ that are of the form $Q\psi$. Note that configurations which are reached by the rules (EQ) and (AQ) applied to $Q\psi$ are "entry points" to parts of the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ in which no more applications of rules (EQ) and (AQ) can be played.
For every $s \in S$ and every $\sigma : \mathit{Var} \to S$ we can solve the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, Q\psi)$ observing that this is a single-player game of at most doubly exponential size whose winning condition is defined by the parity automaton $\mathcal{A}^{Q}_{\mathcal{K},\varphi}$ from Corollary 6.37 of at most doubly exponential size. Parity games in which only one player makes choices boil down to (nested) reachability games which can be solved in NLOGSPACE (Proposition 2.50), hence in NEXPSPACE, i.e. EXPSPACE due to Savitch's Theorem [81] for the games under consideration here. Furthermore, note that the space needed for the additional overhead of enumerating all s and all σ is bounded polynomially.
Then for every σ we take a new proposition $p^{Q\psi}_{\sigma}$ and extend the labelling of $\mathcal{K}$ such that $p^{Q\psi}_{\sigma}$ holds in all those states s such that $\boldsymbol{V}$ wins $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, Q\psi)$.

We then proceed with the next larger subformulas of the form $Q\psi'$ as above but in the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, Q\psi')$ we replace every occurrence of $[Q\psi]^\sigma$ for a previously treated subformula $Q\psi$ with the atomic proposition $p_\sigma^{Q\psi}$. This ensures that these games are single-player games again. This is iterated until all subformulae of φ have been covered. □

Putting the lower bounds from Section 6.1.2 and the upper bound together we obtain completeness.

Corollary 6.39. Model Checking $\mathrm{HCTL}^*_{\mathsf{ps}}$ is ExpSpace-complete, even for expression complexity.

And finally, we obtain that the data complexity of $\mathrm{HCTL}^*_{\mathsf{ps}}$ is still slightly lower.

Corollary 6.40. The data complexity of model checking $\mathrm{HCTL}^*_{\mathsf{ps}}$ is complete for PSpace.

Proof. The lower bound is due to Theorem 6.7. For the upper bound, observe that the game $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ has size at most $|\mathcal{K}| \cdot 2^{|\varphi| \cdot |\mathcal{K}|^k}$ where k is the number of variables used in φ. Thus, for a fixed formula, the game has only exponential size. A similar procedure as in Theorem 6.38 then leads to a PSpace algorithm since the automata used for the winning conditions also only have exponential size. □

The data complexity of $\mathbf{HCTL}^*_{\mathsf{ss}}$. We now come back to the data complexity of $\mathrm{HCTL}^*_{\mathsf{ss}}$ and with it also the data complexity of HCTL, HCTL^+ and HFCTL^+. Remember that it is our goal to achieve an NLogSpace procedure for this problem.
For this we will take another look at the model checking games $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ but this time only for $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$. Remember that in $\mathrm{HCTL}^*_{\mathsf{ss}}$ the binder can only occur as a state formula and not as a genuine path formula as opposed to formulas in $\mathrm{HCTL}^*_{\mathsf{ps}}$. To get some use out of this we first need a new normal form for $\mathrm{HCTL}^*_{\mathsf{ss}}$ formulas.

Definition 6.41. A formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ is in *hybrid normal form* (HNF) if all occurrences of $\downarrow x.\varphi'$ and $@_x\, \varphi$ in φ occur only as part of a prefix directly in front of atomic formulas or $\mathsf{E}\psi$ resp. $\mathsf{A}\psi$ formulas.

For example the formula $\downarrow x.((@_y\, p) \wedge \mathsf{EX}q)$ is not in HNF but the equivalent formula $(\downarrow x.\, @_y\, p) \wedge \downarrow x.(\mathsf{EX}q)$ is. The following Lemma can be proven similarly to Lemma 5.7.

Lemma 6.42. For each formula $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ there is an equivalent formula $\varphi' \in \mathrm{HCTL}^*_{\mathsf{ss}}$ in HNF of length quadratic in the length of φ.

Proof. The proof is easy and straightforward. Since all occurrences of binders and jumps in an $\mathrm{HCTL}^*_{\mathsf{ss}}$ formula are only in front of state formulas we can use the equivalences $\downarrow x.(\varphi_1 \otimes \varphi_2) \equiv \downarrow x.\varphi_1 \otimes \downarrow x.\varphi_2$, $@_x(\varphi_1 \otimes \varphi_2) \equiv @_x\, \varphi_1 \otimes @_x\, \varphi_2$ with $\otimes \in \{\wedge, \vee\}$, $\neg \downarrow x.\varphi \equiv \downarrow x.\neg\varphi$ and $\neg\, @_x\, \varphi \equiv @_x\, \neg\varphi$ to push all hybrid operators downwards over boolean connectives until they are directly over a path quantifier or an atomic formula.
Each application of one of the equivalences above increases the length of the formula by at most one. Further, for each $\downarrow$ or @ we can apply at most length of the formula many of those equivalences and there are at most length of the formula many operators. This immediately gives us the size estimation. $\square$

Without loss of generality we will only consider $\mathrm{HCTL}^*_{\mathsf{ss}}$ formulas in HNF for the data complexity. Since the formula is considered to be fixed the blowup in length that happens while constructing such a formula does not matter.
There are now some key observations we can make about the game rules of the model checking games $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$. First, the rules

$$\frac{s \vdash \mathsf{E}([\downarrow x.\varphi]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}([\varphi]^{\sigma[x \to s]}, \Gamma)} \qquad \frac{s \vdash \mathsf{A}([\downarrow x.\varphi]^{\sigma}, \Gamma)}{s \vdash \mathsf{A}([\varphi]^{\sigma[x \to s]}, \Gamma)}$$

are deterministic in the sense that none of the players makes a choice and thus the application of these rules or the order in which they are applied does not change the winner of a game. Additionally, those rules are the only ones that are able to change the variable assignment of a subformula in a configuration of the game.
Secondly, for formulas in HNF, the binder can only occur in front of atomic formulas p or x, jumps or subformulas $\mathsf{E}\psi$ or $\mathsf{A}\psi$ that create a new path. The game rules associated all three types of formulas – shown here for the case of an outer E path quantifier

$$\frac{s \vdash \mathsf{E}([p]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}(\Gamma)} \text{ if } s \in L(p) \qquad \frac{s \vdash \mathsf{E}([x]^{\sigma}, \Gamma)}{s \vdash_{\sigma} \mathsf{E}(\Gamma)} \text{ if } \sigma(x) = s$$

$$\frac{s \vdash \mathsf{E}([Q\psi]^{\sigma}, \Gamma)}{s \vdash \mathsf{E}(\Gamma) \quad s \vdash Q([\psi]^{\sigma})} \boldsymbol{R} \qquad \frac{s \vdash \mathsf{E}([@_x\varphi]^{\sigma}, \Gamma)}{\sigma(x) \vdash \mathsf{E}([\varphi]^{\sigma}) \quad s \vdash \mathsf{E}(\Gamma)} \boldsymbol{R}$$

all work in a similar way: Either the subformula in question is discarded completely or all other subformulas are discarded completely.
We now combine both observations and restrict the usage of the binder rules for games $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ with φ in HNF such that they can only be applied

directly before one of the four rules above (or the other four rules for the A-quantifier) is applied – essentially compressing the application of the binder rules and the application of one of the four rules above into a single turn. As stated above this change in order does not change the outcome of the game. However, this means that we only have to deal with a single – global – variable assignment for the whole configuration and not with multiple variable assignments for each subformula because the only time the variable assignments is changed with the binder rules happens right before all other subformulas are discarded or the formula with the "changed" variable assignment gets discarded – in which case we do not have to memorise it either.
This means that the model checking games for $\mathcal{G}_{\mathsf{ps}}(\mathcal{K}, s, \sigma, \varphi)$ with φ in HNF can be played on the configuration space $S \times (\mathit{Var} \to S) \times \{\mathsf{E}, \mathsf{A}\} \times 2^{\mathsf{Fl}(\varphi)}$ and thus their size is at most $\mathcal{O}(|\mathcal{K}|^{(k+1)} \cdot 2^{|\varphi|})$ if φ has k variables. To ease the distinction between the normal version and this simplified version we call these games $\mathcal{G}_{\mathsf{ss}}(\mathcal{K}, s, \sigma, \varphi)$
Furthermore, the automaton for the winning condition on infinite A-plays from Theorem 6.36 then also only needs state space $\{\mathsf{E}, \mathsf{A}\} \times 2^{\mathsf{Fl}(\varphi)} \times 2^{\mathcal{U}}$ since it only tracks the satisfaction of the Until-Formulas which now do not need their own variable assignment anymore. Thus, $\mathcal{A}^{\mathsf{E}}_{\varphi,\mathcal{K}}$ for $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ is only of size $\mathcal{O}(2^{|\varphi|})$. The automaton $\mathcal{A}^{\mathsf{A}}_{\varphi,\mathcal{K}}$ from Theorem 6.35 is already only linear in the size of $\mathcal{K}$. The same product constructions with the automata for finite plays as in Corollary 6.37 then give us the following result.

Corollary 6.43. Let $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ be in HNF, let $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure, $s \in S$ and $\sigma : \mathit{Var} \to S$ a variable assignment.
For each $Q \in \{\mathsf{A}, \mathsf{E}\}$ there is a deterministic parity automaton (DPA) $\mathcal{A}^{Q}_{\mathcal{K},\varphi}$ of size exponential in $|\varphi|$ and linear in $|\mathcal{K}|$ which, given an ω-sequence λ, accepts λ iff it is a finite play or an infinite Q-play that is won by player $\boldsymbol{V}$ in $\mathcal{G}_{\mathsf{ss}}(\mathcal{K}, s, \sigma, \varphi)$.

We now have a model checking game that is linear in the size of the Kripke structure as well as automata for the winning conditions that are also only linear in the size of the Kripke structure. This enables us to give an NLogSpace procedure for the data complexity of $\mathrm{HCTL}^*_{\mathsf{ss}}$ model checking.

Theorem 6.44. The data complexity of model checking $\mathrm{HCTL}^*_{\mathsf{ss}}$ is in NLogSpace.

Proof. Without loss of generality, let $\varphi \in \mathrm{HCTL}^*_{\mathsf{ss}}$ be in HNF, $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure and let $\mathcal{G}_{\mathsf{ss}}(\mathcal{K}, s, \sigma, \varphi)$ be the associated model checking game (simplified as described above).

We employ a simple top-down method to solve $\mathcal{G}_{\mathsf{ss}}(\mathcal{K}, s, \sigma, \varphi)$ with the automata from Corollary 6.43 combined with a decomposition method similar to Theorem 6.38.
The lowest parts of the game $\mathcal{G}_{\mathsf{ss}}(\mathcal{K}, s, \sigma, \varphi)$ in which no more applications of rules (EQ) and (AQ) can be played, can be solved in NLogSpace (in the size of $\mathcal{K}$) since they are reachability games and both the game and the automaton for the winning condition are only linear in the size of $\mathcal{K}$.
Parts of the game that are higher up, i.e. there are configurations reachable in which a rule (EQ) and (AQ) can be applied, are solved similarly but with recursive calls to solve the "entry points" to these lower parts of the game. The recursion depth is bounded by the size of the formula which for data complexity is considered fixed. Thus, the algorithm can be implemented to run in NLogSpace. □

Combined with Theorem 6.4 we obtain the following result.

Corollary 6.45. The data complexity for model checking $\mathrm{HCTL}^*_{\mathsf{ss}}$ is NLogSpace-complete.

6.2.3 $\mathbf{HCTL}^*_{\mathsf{pp}}$

We have already seen in Section 6.1.3 that $\mathrm{HCTL}^*_{\mathsf{pp}}$ does not admit an elementary model checking procedure. With this knowledge we can employ powerful machinery like first-order logic that also does not admit an elementary bound.
Again, we will use a decomposition algorithm similar to $\mathrm{HCTL}^*_{\mathsf{ss}}$ and the only difficult case is how to evaluate path formulas. These now feature binder and jump and thus are – at least syntactically – full hybrid LTL formulas. However, even full hybrid LTL can be translated into first-order logic interpreted on paths which in turn can then be translated to Büchi-automata. The basic idea thus is to reduce the model checking problem for $\mathrm{HCTL}^*_{\mathsf{pp}}$ essentially to a series of NBA path problems via a translation of path formulas first into first-order logic and then into Büchi-automata.
However, there is one difficulty that prevents us from simply using the straightforward translation of hybrid LTL into first-order logic: there are slight semantical differences in the interpretation of the binder in hybrid LTL and $\mathrm{HCTL}^*_{\mathsf{pp}}$ that need to be considered.
Hybrid LTL interpreted on paths usually binds a variable to a moment on a path, c.f. [15, 83], and variable tests then do not actually test equality between the stored state and the current one but only test if the *current moment* on the path is the same as the stored one. In $\mathrm{HCTL}^*_{\mathsf{pp}}$ however, we do both: a variable is bound to a moment *and* a state. Jumps then refer to

the moment on the path while variable tests can still test for the actual state in question.
A translation from hybrid LTL to first-order logic is simple and straightforward because variables in first-order logic on infinite paths are – as in hybrid LTL – only interpreted as moments on some path. But this means that there is no actual correspondence for variable tests as in $\mathrm{HCTL}^*_{\mathsf{pp}}$ that try to test for an actual state rather than the moment. This difference needs to be considered and addressed when translating $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas into first-order logic.
We will make these ideas precise by first introducing the notion of *nested existential* NBA *path problems* and will then show how to reduce the model checking problem for $\mathrm{HCTL}^*_{\mathsf{pp}}$ to these nested NBA path problems. The reduction will take a detour over first-order logic addressing the problem raised above.

Nested NBA path problems.

Definition 6.46. The set of *nested existential* NBA *path problem expressions* – or short nENBA expressions – is the least set that is generated by the following grammar:

$$\mathsf{exp} \;:=\; p \mid \mathsf{exp} \vee \mathsf{exp} \mid \neg\mathsf{exp} \mid {\downarrow} x.\mathsf{exp} \mid @_x\, \mathsf{exp} \mid \mathsf{E}\mathcal{A}$$

where $p \in Prop \cup Var$ and $\mathcal{A}$ is an NBA over $Prop \cup Var$ and a fresh set of atomic propositions $\{p_{exp_1}, \ldots, p_{exp_n}\}$ where $exp_i \in$ nENBA for all $i = 1, \ldots, n$.

The satisfaction relation is given inductively based on the semantics for hybrid branching-time logics coupled with the idea of the decomposition algorithm for $\mathrm{HCTL}^*_{\mathsf{ss}}$. Variable assignments are directly encoded into the Kripke structure so that the Büchi-automaton can check variable tests exactly like normal propositions and will be updated at appropriated steps. This was already defined in Definition 5.21. Remember that by this definition $\mathcal{K}_\sigma$ denotes the structure where the labelling L has been extended with $L(x) = \{\sigma(x)\}$ for all $x \in Var$.
Each nENBA expression is evaluated with respect to a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ over $Prop \cup Var$ and a state $s \in S$.

$$\begin{aligned}
\mathcal{K}, s \models p \text{ iff } &\quad s \in L(p),\\
\mathcal{K}, s \models exp_1 \vee exp_2 \text{ iff } &\quad \mathcal{K}, s \models exp_1 \text{ or } \mathcal{K}, s \models exp_2,\\
\mathcal{K}, s \models \neg exp \text{ iff } &\quad \mathcal{K}, s \not\models exp,\\
\mathcal{K}, s \models {\downarrow} x.exp \text{ iff } &\quad \mathcal{K}', s \models exp \text{ with } \mathcal{K}' = \mathcal{K} \cup \{\sigma[x \mapsto s]\},
\end{aligned}$$

$$\begin{array}{rl} \mathcal{K}, s \models @_x\, exp \text{ iff} & \mathcal{K}, s_x \models exp \text{ where } s_x \text{ is the unique state} \\ & \text{with } s_x \in L(x), \\ \mathcal{K}, s \models \mathsf{E}\mathcal{A} \text{ iff} & \text{there is a path } \pi \text{ starting at } \pi^0 = s \text{ on } \mathcal{K}' \text{ accepted} \\ & \text{by } \mathcal{A} \text{ where } \mathcal{K}' = \langle S, \rightarrow, L' \rangle \text{ with } L'(p) = L(p) \\ & \text{for all propositions } p \text{ occurring in } \mathcal{K} \text{ and} \\ & L(p_{exp_i}) = \{s \in S \mid \mathcal{K}, s \models exp_i\}. \end{array}$$

The *nested existential NBA problem* – or short nENBA problem – is the following:

Input: a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L \rangle$, a state $s \in S$ and an nENBA *exp*.

Output: Does $\mathcal{K}, s \models exp$ hold?

The nested existential NBA problem is a generalisation of the existential NBA path problem.

Definition 6.47. The size of an nENBA *exp* or simply $|exp|$ is straightforwardly defined as:

$$\begin{array}{rcl} |p| & := & 1, \\ |exp_1 \vee exp_2| & := & |exp_1| + |exp_2| + 1, \\ |\neg exp| & := & |exp| + 1, \\ |{\downarrow} x.exp| & := & |exp| + 1, \\ |\,@_x\, exp| & := & |exp| + 1, \\ |\mathsf{E}\mathcal{A}| & := & |\mathcal{A}| + \sum_{i=1}^{n} |exp_i| \end{array}$$

where $\mathcal{A}$ is an NBA using atomic propositions $p_{exp_1}, \ldots, p_{exp_n}$ with $exp_i \in$ nENBA for $i = 1, \ldots, n$.

The following lemma about nested existential NBA problems is easy to see.

Lemma 6.48. The nENBA path problem on a Kripke Structure $\mathcal{K}$ and an nENBA *exp* is solvable in space $\mathcal{O}(\max\{|K|, (log(|exp|))^2\})$.

Proof. nENBA can be solved in the same way as $\text{HCTL}^*_{\mathsf{ss}}$ formulas with a decomposition algorithm similar to Algorithm 1 described for $\text{HCTL}^*_{\mathsf{ss}}$ model checking. The only difference is how path formulas, or in our case now

Büchi automata, are treated. For this we first recursively solve the nENBA-subexpressions in a Büchi automaton and adjust the labels of the states accordingly taking space $\mathcal{O}(|K|)$. Then, instead of invoking an LTL model checker we have to solve an existential NBA path problem which according to Proposition 2.38 can be solved in NLOGSPACE and therefore in deterministic space $\mathcal{O}(log(|exp|)^2)$ [81]. □

We will reduce the model checking problem of $\mathrm{HCTL}^*_{\mathsf{pp}}$ to the nENBA path problem. In essence, nENBA are $\mathrm{HCTL}^*_{\mathsf{pp}}$ state formulas where the path formulas have been replaced by (nested) Büchi automata. Thus, we need to explain how to construct a Büchi-automaton for a path formula $\mathsf{E}\psi \in \mathrm{HCTL}^*_{\mathsf{pp}}$.
We do this in two steps. First, we will translate a path formula into a first-order formula on ω-words and second, we will construct an equivalent Büchi-automaton for this FO formula. Depending on the quantifier alternation this Büchi-automaton may be quite large. This is no suprise, because we know from Section 6.1.3 already that there cannot be an elementary decision procedure for $\mathrm{HCTL}^*_{\mathsf{pp}}$. Hence, there cannot be an elementary bound for these Büchi-automata.
The construction of a Büchi-automaton from an FO formula is well-known [18]. However translating an $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formula into an equivalent FO formula is not so simple as discussed already at the beginning of this section.
Remember that in $\mathrm{HCTL}^*_{\mathsf{pp}}$ bound variables along a path essentially have two meanings:

1. For variable tests x we need to store the actual *state* to check if the current state of the evaluation is actually the same state that was stored, while

2. for jumps we store the *moment* of the path so that we can jump back to this exact moment along the path.

For FO formulas on infinite words, variables are typically interpreted as moments along an infinite path, hence it is easy to simulate the second part, i.e. jumps to some moment. However, there is no actual correspondence to storing the actual states along this infinite path.
Since we are only interested in a procedure for model checking, we can employ a simple trick. Instead of model checking on the actual structure $\mathcal{K}$ we enhance this structure with fresh nominals for each state of the structure. These nominals then identify a single state and can then be used to simulate the variable tests.

Definition 6.49. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure and let Nom_S be a fresh set of nominals not present in $\mathcal{K}$ with $|Nom_S| = |S|$. Since there is exactly one fresh nominal for each state we assume that each state s is associated with one new nominal and call this nominal n_s.
We define $\mathcal{K}^S := \langle S, \rightarrow, L'\rangle$ as the *enhanced Kripke structure* such that $L'(p) = L(p)$ for all $p \in Prop \cup Nom$ present in $\mathcal{K}$ and $L(n_s) = \{s\}$ for every $n_s \in Nom_S$.

Thus, every state in an enhanced Kripke structure is additionally labelled with its own unique nominal. The following fact about enhanced structures and formulas that do not use the newly added vocabulary follows immediately.

Proposition 6.50. Let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure. For every $\mathrm{HCTL}^*_{\mathsf{pp}}$ formula φ that does not have any occurrence of nominals from Nom_S and every $s \in S$, $\sigma : Var \rightarrow S$ it holds that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}^S, s, \sigma \models \varphi$.

We will now introduce FO on infinite paths before giving the translation from $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas to FO formulas.
Note first that any Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$ can be regarded as a first-order structure with universe S over the signature $\tau = \langle R^{(2)}, Prop^{(1)} \cup Nom^{(1)}\rangle$ with a binary accessibility relation R and unary predicates for each $p \in Prop$ and $n \in Nom$. For enhanced Kripke structures we assume that they are interpreted over the signature $\tau = \langle R^{(2)}, Prop^{(1)} \cup Nom^{(1)} \cup Nom_S^{(1)}\rangle$ with special nominals $n_s \in Nom_S$ for each state.
FO formulas over the signature τ are then given by the grammar

$$\varphi \ := \ x < y \mid R(x,y) \mid p(x) \mid n(x) \mid n_s(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi$$

with $x, y \in Var$. The usual abbreviations like $x \leq y := \neg(y < x)$, $\forall x.\varphi := \neg\exists x.\neg\varphi$ etc. are used freely.
Formulas of FO are interpreted over infinite paths in (enhanced) Kripke structures. So, let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure, π a path in $\mathcal{K}$ and $\vartheta : Var \rightarrow \mathbb{N}$ a variable interpretation. The satisfaction relation is given inductively:

$$\begin{aligned}
\mathcal{K}, \pi, \vartheta \models x < y \text{ iff} &\quad \vartheta(x) < \vartheta(y),\\
\mathcal{K}, \pi, \vartheta \models R(x,y) \text{ iff} &\quad \pi^{\vartheta(x)} \rightarrow \pi^{\vartheta(y)} \text{ in } \mathcal{K},\\
\mathcal{K}, \pi, \vartheta \models p(x) \text{ iff} &\quad \pi^{\vartheta(x)} \in L(p),\\
\mathcal{K}, \pi, \vartheta \models n(x) \text{ iff} &\quad \pi^{\vartheta(x)} \in L(n),
\end{aligned}$$

$$\begin{aligned}
\mathcal{K},\pi,\vartheta \models \neg\varphi \text{ iff} &\quad \mathcal{K},\pi,\vartheta \not\models \varphi,\\
\mathcal{K},\pi,\vartheta \models \varphi_1 \vee \varphi_2 \text{ iff} &\quad \mathcal{K},\pi,\vartheta \models \varphi_1 \text{ or } \mathcal{K},\pi,\vartheta \models \varphi_2,\\
\mathcal{K},\pi,\vartheta \models \exists x.\varphi \text{ iff} &\quad \text{there is } k \in \mathbb{N} \text{ such that } \mathcal{K},\pi,\vartheta[x \mapsto k] \models \varphi.
\end{aligned}$$

The usual abbreviations for universal quantification, implications etc. are freely used.
We will now present the translation from $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas to FO formulas.
Translations from modal, temporal and hybrid logics to first-order logic are usually given with respect to a free variable z interpreted as the "current state" in the evaluation of a modal/temporal/hybrid formula. Propositions p and nominals n are interpreted as unary predicates.
In order to check for the correct states along a path with variable tests our translation will also be relative to a variable interpretation $\sigma : \mathit{Var} \to S$. This helps us to store the states and use the correct (newly added) nominals n_s for a state $s \in S$ in case of variable tests as discussed above.
Let ψ, ψ_1, ψ_2 be $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas such that all maximal state subformulas are only propositions or nominals. Note that free variables in these path formulas can be seen as nominals in the context of this path formula and thus are not considered here. However, there may be variable tests for variables that have been bound during the context of such a path formula.
The translation of such a path formula ψ is given by mutual recursion on the structure of ψ:

$$\begin{aligned}
\mathsf{tr}^\sigma_z(p) &:= p(z),\\
\mathsf{tr}^\sigma_z(n) &:= n(z),\\
\mathsf{tr}^\sigma_z(x) &:= (n_{\sigma(x)})(z),\\
\mathsf{tr}^\sigma_z(\neg\psi) &:= \neg\mathsf{tr}^\sigma_z(\psi),\\
\mathsf{tr}^\sigma_z(\psi_1 \vee \psi_2) &:= \neg\mathsf{tr}^\sigma_z(\psi_1) \vee \mathsf{tr}^\sigma_z(\psi_2),\\
\mathsf{tr}^\sigma_z(\mathsf{X}\psi) &:= \exists y.R(z,y) \wedge \mathsf{tr}^\sigma_y(\psi),\\
\mathsf{tr}^\sigma_z(\psi_1 \mathsf{U} \psi_2) &:= \exists y.y \geq z \wedge \mathsf{tr}^\sigma_y(\psi_2) \wedge \forall x.(y > x \wedge x \geq z) \to \mathsf{tr}^\sigma_x(\psi_1),\\
\mathsf{tr}^\sigma_z(\downarrow x.\psi) &:= \bigwedge_{s \in S} (n_s)(z) \to \mathsf{tr}^{\sigma[x \mapsto s]}_z(\psi),\\
\mathsf{tr}^\sigma_z(@_x\, \psi) &:= \mathsf{tr}^\sigma_x(\psi),
\end{aligned}$$

where $x \in \mathit{Var}$ is a variable that was bound beforehand in the context of the path formula. For a $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formula ψ the initial variable assignment σ is irrelevant since it is only used to remember the correct nominals for variables that get bound on the path anyways. We thus simply write $\mathsf{tr}_z(\psi)$ for the translation of ψ with regard to the free variable z.

Note that as discussed earlier, the binder and variable tests use the newly introduced nominals in the enhanced structure to store and check the current state.

Lemma 6.51. Let ψ be a $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formula, possibly containing free variables, such that there are only maximal state subformulas of the form $p \in \mathit{Prop}$, $n \in \mathit{Nom}$, $x \in \mathit{Var}$ and let $\mathcal{K} = \langle S, \rightarrow, L\rangle$ be a Kripke structure and $\mathcal{K}^S$ its enhanced structure. Additionally, let $z \in \mathit{Var}$ be a variable that does not occur in ψ. Then

$$\pi, k, \sigma, \vartheta \models \psi \text{ if and only if } \pi, \vartheta' \models \mathsf{tr}^{\sigma}_{z}(\psi),$$

for all paths π in $\mathcal{K}^S$, $k \in \mathbb{N}$, $\sigma : \mathit{Var} \rightarrow S$, $\vartheta : \mathit{Var} \rightarrow \mathbb{N}$ and $\vartheta' : \mathit{Var} \rightarrow \mathbb{N}$ with $\vartheta'(z) = k$ and $\vartheta'(y) = \vartheta(y)$ for all $y \in \mathsf{free}(\psi)$.

Proof. We prove this by induction on the structure of ψ. So, let π be a path in $\mathcal{K}^S$, $k \in \mathbb{N}$, $\sigma : \mathit{Var} \rightarrow S$, $\vartheta, \vartheta' : \mathit{Var} \rightarrow \mathbb{N}$ with $\vartheta'(z) = k$ and $\vartheta'(y) = \vartheta(y)$ for all $y \in \mathsf{free}(\psi)$.
"$\Rightarrow$" Assume that $\pi, k, \sigma, \vartheta \models \psi$. For the base cases:

- If $\psi = p$, then $\pi^k \in L(p)$ and since $\vartheta'(z) = k$ we also have that $\pi^{\vartheta'(z)} \in L(p)$ and consequently $\pi, \vartheta' \models p(z)$. The case for nominals is basically the same.
- If $\psi = x$, then $\pi^k = \sigma(x)$ by the semantics of $\mathrm{HCTL}^*_{\mathsf{pp}}$ formula which means that $\pi^k \in L(n_{\sigma(x)})$ since $n_{\sigma(x)} \in \mathit{Nom}_S$ only holds at state $\sigma(x)$. And from this it follows with $\vartheta'(z) = k$ that $\pi, \vartheta \models n_{\sigma(x)}(z)$.

The cases for $\psi = \neg\psi_1$ and $\psi = \psi_1 \vee \psi_2$ follow immediately. For the other cases of temporal and hybrid operators consider the following:

- If $\psi = \mathsf{X}\psi_1$, then $\pi, k+1, \sigma, \vartheta \models \psi_1$. By the induction hypothesis we get that $\pi, \vartheta'' \models \mathsf{tr}^{\sigma}_{y}(\psi_1)$ for any $\vartheta'' : \mathit{Var} \rightarrow \mathbb{N}$ such that $\vartheta''(y) = k+1$ and $\vartheta''(x) = \vartheta(x)$ for all $x \in \mathsf{free}(\psi_1)$. Thus, since $\mathsf{free}(\psi) = \mathsf{free}(\psi_1)$ and $\vartheta'(z) = k$ we get that $\pi, \vartheta' \models \exists y. y = z + 1 \wedge \mathsf{tr}^{\sigma}_{y}(\psi_1)$.
- If $\psi = \psi_1 \mathsf{U} \psi_2$ and $\pi, k, \sigma \models \psi$, then there is some $k' \geq k$ such that for all $k \leq j < k'$ it holds that $\pi, j, \sigma \models \psi_1$ and $\pi, k', \sigma \models \psi_2$. By the induction hypothesis we thus get that $\pi, \vartheta'' \models \mathsf{tr}^{\sigma}_{x}(\psi_1)$ for all ϑ'' such that $\vartheta''(x) = j$ and $\vartheta''(y) = \vartheta(y)$ for all $y \in \mathsf{free}(\psi_1) \subseteq \mathsf{free}(\psi)$. And we also get that $\pi, \vartheta''' \models \mathsf{tr}^{\sigma}_{y}(\psi_2)$ for all ϑ''' such that $\vartheta'''(y) = k'$ and $\vartheta'''(x) = \vartheta(x)$ for all $x \in \mathsf{free}(\psi_2) \subseteq \mathsf{free}(\psi)$. Thus, it holds that $\pi, \vartheta' \models \exists y. y \geq z \wedge \mathsf{tr}^{\sigma}_{y}(\psi_2) \wedge \forall x.(y > x \wedge x \geq z) \rightarrow \mathsf{tr}^{\sigma}_{x}(\psi_1)$.

- If $\psi = {\downarrow} x.\psi_1$ then $\pi, k, \sigma[x \mapsto \pi^k] \models \psi_1$. By the induction hypothesis we get that $\pi, \vartheta'' \models \mathsf{tr}_z^{\sigma[x\mapsto\pi^k]}(\psi_1)$ for all ϑ'' with $\vartheta''(z) = k$ and $\vartheta''(x) = \vartheta(x)$ for all $x \in \mathsf{free}(\psi_1)$. Especially we have that $\pi, \vartheta' \models \mathsf{tr}_z^{\sigma[x\mapsto\pi^k]}(\psi_1)$. Let $\pi^k = s$ for some state $s \in S$. Then there is exactly one nominal $n_s \in \mathit{Nom}_S$ such that $\pi^k \in L(n_s)$, resp. $\pi, \vartheta' \models n_s(z)$. And hence $\pi, \vartheta' \models \bigwedge_{s\in S}(n_s)(z) \to \mathsf{tr}_z^{\sigma[x\mapsto s]}(\psi_1)$.

- If $\psi = @_x\,\psi_1$, then $\pi, \vartheta(x), \sigma, \vartheta \models \psi_1$. Thus, by the induction hypothesis we get that $\pi, \vartheta'' \models \mathsf{tr}_y^\sigma(\psi_1)$ for any $\vartheta'' : \mathit{Var} \to \mathbb{N}$ such that $\vartheta''(y) = \vartheta(x)$ and $\vartheta''(z) = \vartheta(z)$ for all $z \in \mathsf{free}(\psi_1)$.

  We now have two cases. First, suppose that $x \in \mathsf{free}(\psi_1)$. If this is the case then we have that $\vartheta''(y) = \vartheta(x) = \vartheta''(x)$ by combining both conditions. And since x and y agree on their interpretation in ϑ'' we can substitute y by x in $\mathsf{tr}_y^\sigma(\psi_1)$, thus achieving that $\pi, \vartheta'' \models \mathsf{tr}_x^\sigma(\psi_1) = \mathsf{tr}_y^\sigma(\psi_1)\big[x/y\big]$. And since y does not occur anymore in $\mathsf{tr}_x^\sigma(\psi_1)$, its assignment in ϑ'' becomes irrelevant and thus, we also get that $\pi, \vartheta' \models \mathsf{tr}_x^\sigma(\psi_1)$.

  Secondly, suppose that $x \notin \mathsf{free}(\psi_1)$ meaning that there are no jumps or variable tests in ψ_1 involving x. Then in $\mathsf{tr}_y^\sigma(\psi_1)$ the variable x is not used and we can simply rename y and we get that for all variable assignments ϑ'' with $\vartheta''(x) = \vartheta(x)$ and $\vartheta''(z) = \vartheta(z)$ for all $z \in \mathsf{free}(\psi_1)$ we have that $\pi, \vartheta'' \models \mathsf{tr}_x^\sigma(\psi_1)$. Thus, especially we have that $\pi, \vartheta' \models \mathsf{tr}_x^\sigma(\psi_1)$.

This finishes the "⇒" direction. The reverse direction ("⇐") follows completely analogously with the same arguments. □

The following fact for closed path formulas follows directly.

Corollary 6.52. Let $\mathsf{E}\psi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ such that there are only maximal state subformulas of the form $p \in \mathit{Prop}$, $n \in \mathit{Nom}$ and let $\mathcal{K} = \langle S, \to, L\rangle$ be a Kripke structure and $\mathcal{K}^S$ its enhanced structure. Finally, let $z \in \mathit{Var}$ be a variable that does not occur in $\mathsf{E}\psi$. Then

$$\pi, k \models \psi \text{ if and only if } \pi, \vartheta \models \mathsf{tr}_z(\psi),$$

for all paths π in $\mathcal{K}^S$, $k \in \mathbb{N}$ and $\vartheta : \mathit{Var} \to \mathbb{N}$ such that $\vartheta(z) = k$.

It is well-known that first-order formulas can be translated into Büchi-automata.

Proposition 6.53 ([18])**.** For each FO formula on infinite paths φ of size n and alternation depth $\mathsf{ad}(\varphi) = m$ there is an NBA $\mathcal{A}_\varphi$ of size at most 2^n_m such that for all paths π, π is accepted by $\mathcal{A}_\varphi$ if and only if $\pi \models \varphi$.

We now have all the tools together to solve the model checking problem for $\mathrm{HCTL}^*_{\mathsf{pp}}$. We solve this by reducing it to the nENBA path problem.

Theorem 6.54. For each $\mathrm{HCTL}^*_{\mathsf{pp}}$ formula φ, there is an nENBA expression e_φ such that for all Kripke structures $\mathcal{K} = \langle S, \rightarrow, L \rangle$, states $s \in S$ and variable assignments $\sigma : \mathit{Var} \rightarrow S$ it holds that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}^S_\sigma, s \models e_\varphi$.

Proof. Let $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ and $\mathcal{K} = \langle S, \rightarrow, L \rangle$ be a Kripke structure. Without loss of generality we assume that φ does not use nominals from Nom_S. Then by Proposition 6.50 we get that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}^S, s, \sigma \models \varphi$.
We first construct an nENBA expression e_φ inductively on the structure of $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ and show correctness of the construction afterwards.
The cases for propositions, boolean combinations and hybrid operators on state formulas simply remain unchanged since nENBA expressions also feature these constructs. Thus for state formulas $\varphi, \varphi_1, \varphi_2 \in \mathrm{HCTL}^*_{\mathsf{pp}}$ we get:

$$\begin{aligned}
\tau(p) &= p,\\
\tau(n) &= n,\\
\tau(x) &= x,\\
\tau(\neg\varphi) &= \neg\tau(\varphi),\\
\tau(\varphi_1 \vee \varphi_2) &= \tau(\varphi_1) \vee \tau(\varphi_2),\\
\tau(@_x\, \varphi) &= @_x\, \tau(\varphi),\\
\tau({\downarrow} x.\varphi) &= {\downarrow} x.\tau(\varphi).
\end{aligned}$$

This only leaves us with the case for $\mathsf{E}\psi$ for some path formula ψ. As often before, let $\varphi_1, \dots, \varphi_n$ be the maximal state-subformulas of ψ for some $n \in \mathbb{N}$. We first translate $\varphi_1, \dots, \varphi_n$ recursively into nENBA expressions. Let $e_1, \dots, e_n \in \mathsf{nENBA}$ be their respective translations.
Let $\psi' = \psi[p_{e_1}/\varphi_1, \dots, p_{e_n}/\varphi_n]$ be the formula derived from ψ in which all occurrences of φ_i are replaced by atomic propositions p_{e_i} for $i = 1, \dots, n$. Using Corollary 6.52 we first construct an FO formula $\mathsf{tr}_z(\psi')$ and with Proposition 6.53 we then construct a Büchi-automaton $\mathcal{A}_{\mathsf{tr}_z(\psi')}$ over the additional atomic propositions $p_{e_1}, \dots, p_{e_n}$ for the nENBA expressions $e_1, \dots, e_n$. Then $\tau(\mathsf{E}\psi) = \mathsf{E}\mathcal{A}_{\mathsf{tr}_z(\psi')}$.
We then simply use $e_\varphi := \tau(\varphi)$. It remains to show correctness of the construction, i.e. $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K}^S_\sigma, s \models e_\varphi$.
For this, assume that $\mathcal{K}, s, \sigma \models \varphi$. For $\varphi = p$ we get that $\mathcal{K}^S_\sigma, s \models p$ since $\mathcal{K}$ and $\mathcal{K}^S_\sigma$ agree on all atomic propositions. The same holds for $\varphi = n$ for some $n \in \mathit{Nom}$.
For $\varphi = x$ we get that $s = \sigma(x)$ from $\mathcal{K}, s, \sigma \models x$ and thus by construction of $\mathcal{K}^S_\sigma$ we have that $s \in L'(x)$ and thus $\mathcal{K}^S_\sigma, s \models x$. The cases for negation and disjunction follow immediately.

For $\varphi = @_x \varphi_1$ it follows that $\mathcal{K}, \sigma(x), \sigma \models \varphi_1$ and thus by assumption we get that $\mathcal{K}^S_\sigma, \sigma(x) \models \tau(\varphi_1)$. By construction of $\mathcal{K}^S_\sigma$, we have that $\sigma(x)$ is the unique state with $\sigma(x) \in L'(x)$ and thus $\mathcal{K}^S_\sigma, s \models @_x \tau(\varphi_1)$.

For $\varphi = \downarrow x.\varphi_1$ it follows that $\mathcal{K}, s, \sigma[x \mapsto s] \models \varphi_1$ and by assumption we get that $\mathcal{K}^S_{\sigma[x \mapsto s]}, s \models \tau(\varphi_1)$ and immediately by the semantics of nENBA expressions we have that $\mathcal{K}^S_\sigma, s \models \downarrow x.\tau(\varphi_1)$.

Lastly, assume that $\varphi = \mathsf{E}\psi$. By Proposition 6.50 we get that $\mathcal{K}^S, s, \sigma \models \mathsf{E}\psi$. Let $(\mathcal{K}^S)'$ be the Kripke structure that extends the labeling of $\mathcal{K}^S$ with $L(p_{e_i}) = \{s \in S \mid \mathcal{K}^S, s, \sigma \models \varphi_i\}$ for the maximal state-subformulas $\varphi_1, \ldots, \varphi_n$ of ψ.

It can be shown by a straightforward induction that for all $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ it holds that $\mathcal{K}^S, s, \sigma \models \varphi$ if and only if $(\mathcal{K}^S)', s, \sigma \models \varphi[p_{e_1}/\varphi_1, \ldots, p_{e_n}/\varphi_n]$. Thus, we especially get that $(\mathcal{K}^S)', s, \sigma \models \mathsf{E}\psi'$.

Lastly, we need to deal with the possible free variables in ψ'. As mentioned at the beginning of Section 3.2.1 we can regard free variables simply as nominals. As a consequence we have for all formulas $\varphi \in \mathrm{HCTL}^*_{\mathsf{pp}}$ with possible free variables that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\mathcal{K} \cup \{\sigma\}, s \models \varphi$. Thus, we especially get that $(\mathcal{K}^S)' \cup \{\sigma\}, s \models \mathsf{E}\psi'$ by simply regarding all free occurrences of variables as nominals.

By the semantics of $\mathrm{HCTL}^*_{\mathsf{pp}}$ this means that there is a path π in $(\mathcal{K}^S)' \cup \{\sigma\}$ starting at s such that $\pi, 0 \models \psi'$. Using Corollary 6.52 we get that $\pi, \vartheta \models \mathsf{tr}_z(\psi')$ for all $\vartheta : \mathit{Var} \to \mathbb{N}$ with $\vartheta(z) = 0$, i.e. $\pi \models \mathsf{tr}_z(\psi')$. And by Proposition 6.53 we have that π is accepted by $\mathcal{A}_{\mathsf{tr}_z(\psi')}$ and thus $\mathcal{K}^S_\sigma, s \models \mathsf{E}\mathcal{A}_{\mathsf{tr}_z(\psi')}$ by the semantics of nENBA expressions. $\square$

Thus, using Lemma 6.48, we can solve the model checking problem by solving its equivalent nENBA. The translation from $\mathrm{HCTL}^*_{\mathsf{pp}}$ formulas to nENBA expressions is linear except for the case of $\mathsf{E}\psi$ formulas. For such formulas, the translation to FO is linear, however the temporal operator U creates a nesting of a universal quantifier underneath an existential one. Thus, the quantifier alternation depth of the resulting FO formula is bounded by $2n$ where n is the size of the formula that gets translated. By Proposition 6.53 the Büchi automaton constructed from this formula is bounded by 2^n_{2n} and thus the size of the resulting nENBA is bounded by $\mathcal{O}(2^n_{2n})$. With Lemma 6.48 such an nENBA expression can be solved in $2n$−ExpSpace.

This estimate shows that the model checking problem for $\mathrm{HCTL}^*_{\mathsf{pp}}$ formulas of size n can be solved in $2n$-ExpSpace. However, a small example of the translation from path formulas into FO shows that this is only a very rough estimation.

Example 6.55. Consider the two families of formulas

$$\begin{aligned}\varphi_n &:= \mathsf{E}(p_1\mathsf{U}(p_2\mathsf{U}(p_3\mathsf{U}(\cdots\mathsf{U}(p_n\mathsf{U}p_{n+1})\ldots), \text{ and}\\ \varphi'_n &:= \mathsf{E}(\ldots((p_1\mathsf{U}p_2)\mathsf{U}p_3)\mathsf{U}\ldots)\mathsf{U}p_{n+1}).\end{aligned}$$

The difference between these two families is that for φ_n the nesting of the Until formulas is only on the right-hand side of the Until formulas, while for φ'_n the nesting is to the left-hand side of the Until operator.
Now consider the FO translation of the embedded path formulas. For φ_n we obtain relative to an initial variable z the FO formula

$$\begin{aligned}&\exists y_1.y_1 \geq z \wedge \exists y_2.y_2 \geq y_1 \wedge \ldots \wedge \exists y_n.y_n \geq y_{n-1} \wedge p_{n+1}(y_n)\wedge\\ &\quad(\forall x.(y_n > x \geq y_{n-1}) \rightarrow p_n(x))\wedge\\ &\quad(\forall x.(y_{n-1} > x \geq y_{n-2}) \rightarrow p_{n-1}(x))\wedge\\ &\quad\vdots\\ &\quad(\forall x.(y_1 > x \geq z) \rightarrow p_1(x)).\end{aligned}$$

Clearly, these formulas have a constant quantifier alternation depth for all $n \in \mathbb{N}$. On the other hand we obtain

$$\begin{aligned}\exists y_n.y_n \geq z \wedge p_{n+1}(y_n) \wedge \big(\forall x.(y_n > x \geq z) \rightarrow&\\ &\exists y_{n-1}.y_{n-1} \geq x \wedge p_n(y_{n-1})\wedge\\ &\big(\forall z.(y_{n-1} > z \geq x) \rightarrow \ldots\end{aligned}$$

for the formulas φ'_n. And as we can see, the quantifier alternation depth of these formulas is $2(n-1)$.
Thus, the Büchi-automata obtained for φ_n are magnitudes smaller than those obtained for φ'_n.

To obtain a more precise picture of the complexity for $\mathrm{HCTL}^*_{\mathsf{pp}}$ model checking we thus present a more precise measure for $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas that captures the alternation depth of its equivalent FO formula.

Definition 6.56. For all $i \in \mathbb{N}$ we define the classes Σ_i, Π_i of $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas as follows:

1. $\Sigma_0 = \Pi_0$ contains all purely propositional $\mathrm{HCTL}^*_{\mathsf{pp}}$ path formulas.
2. $\Pi_i \cup \Sigma_i \subseteq \Sigma_{i+1}$ and $\Pi_i \cup \Sigma_i \subseteq \Pi_{i+1}$.
3. If $\varphi_1, \varphi_2 \in \Sigma_i$, then also $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \mathsf{X}\varphi_1, \downarrow x.\varphi_1, @_x\,\varphi_1 \in \Sigma_i$ for all $x \in \mathit{Var}$. The same is true for Π_i.

4. If $\varphi \in \Sigma_i$ then $\neg\varphi \in \Pi_i$ and vice versa.

5. If $\psi_1 \in \Pi_j$ and $\psi_2 \in \Sigma_h$ with $j \leq i$ and $h \leq i+1$ then $\psi_1 \mathsf{U} \psi_2 \in \Sigma_{i+1}$.

Let ψ be an HCTL$^*_{\mathsf{pp}}$ path formula and let i be the smallest index, such that $\psi \in \Sigma_i$. We then say that the *alternation depth of* ψ is i.
Let $\varphi \in$ HCTL$^*_{\mathsf{pp}}$ and let i be the biggest quantifier alternation depth of all its path subformulas. Then the *alternation depth of* φ is i.

Intuitively, Σ_i captures the HCTL$^*_{\mathsf{pp}}$ path formulas whose FO translations have at most alternation depth i with an existential quantifier as the topmost quantifier in its syntax tree. Π_i captures those with alternation depth i and a universal quantifier as the topmost one. Notice however that our translation as introduced above does not introduce universal quantifiers as the topmost operator in any translation. Hence, the definition above is leaning almost entirely to the Σ_i case.
At first glance, the fact that Σ_i and Π_i are closed under X may seem contradictory to the intuition given in the paragraph above because the translation of a Next-operator introduces a new existential quantifier. Thus, formulas of the form $\mathsf{X}\varphi$ with $\varphi \in \Pi_i$ should be in Σ_{i+1} and not in Π_i anymore. However, the translation above is only one possibility to translate the Next-operator, another being $\mathsf{tr}^\sigma_z(\mathsf{X}\psi) := \forall y. y = z + 1 \rightarrow \mathsf{tr}^\sigma_y(\psi)$. Switching between both possibilities at appropriate times enables us to be even more precise when measuring the complexity.
Finally, notice that the fourth clause of Until-formulas truely covers all cases (when combined with the second clause). For example, if $\psi_1 \in \Sigma_j$, then by the second case $\psi_1 \in \Pi_{j+1}$ and we can use the fourth clause. Notice further that if $\psi_1 \in \Sigma_j$, the alternation depth of $\mathsf{tr}^\sigma_z(\psi_1 \mathsf{U} \psi_2)$ would increase by at least 2, exactly as it happens here. The same can easily be checked for the other cases as well. This observation gives us the following fact which can easily be checked.

Proposition 6.57. Let ψ be an HCTL$^*_{\mathsf{pp}}$ path formula with alternation depth n. Then for any Kripke structure $\mathcal{K} = \langle S, \rightarrow, L \rangle$ and any $\sigma : \mathit{Var} \rightarrow S$ the first-order quantifier alternation depth of $\mathsf{tr}^\sigma_{\text{¸}}(\psi)$ is n.

With this we are now able to give a precise upper bound for the model checking procedure of HCTL$^*_{\mathsf{pp}}$.

Theorem 6.58. Model Checking for HCTL$^*_{\mathsf{pp}}$ formulas with alternation depth n is in n-ExpSpace.

Proof. To show this, notice that the nENBA expression for an $\text{HCTL}^*_{\text{pp}}$ formula φ with alternation depth n is bounded by $\mathcal{O}(2^{|\varphi|}_n)$ since the translation from $\text{HCTL}^*_{\text{pp}}$ path formulas to FO is polynomial in the size of the formula and the alternation depth of the resulting FOformula is n. Thus, by Proposition 6.53 the resulting Büchi automata are bounded by $\mathcal{O}(2^{|\varphi|}_n)$ and thus the nENBA expression as well. By Lemma 6.48 the path problem for this nENBA can be solved in n-EXPSPACE. □

This upper bound also holds when considering a fixed structure since the complexity originates from the size of the constructed Büchi automata which solely depend on the formula.

Corollary 6.59. The expression complexity for $\text{HCTL}^*_{\text{pp}}$ formulas with quantifier alternation depth n is in n-EXPSPACE.

For a fixed formula we get a better complexity. Perhaps surprisingly, the complexity even drops to the level of $\text{HCTL}^*_{\text{ps}}$ model checking with a fixed formula.

Theorem 6.60. The data complexity for $\text{HCTL}^*_{\text{pp}}$ is PSPACE-complete.

Proof. The lower bound can be transferred from $\text{HCTL}^*_{\text{ps}}$ model checking in Theorem 6.7. For the upper bound, notice that for a fixed formula, the NBA constructed in the process are of constant size. Thus, $\text{HCTL}^*_{\text{pp}}$ model checking can be reduced to an nENBA path problem with NBA of fixed size. According to Lemma 6.48 this is solvable in PSPACE. □

6.2.4 The Fully Hybrid μ-calculus H_μ

As seen in Section 6.1.4, model checking for H_μ is at least EXPTIME-hard. We will now give a matching upper bound by reducing the H_μ model checking problem to the model checking problem of L_μ with multiple modalities.
The multi-modal μ-calculus is defined in the same way as the μ-calculus in Section 2.4 with the exception that instead of only $\Diamond\varphi$ we now have multiple modalities resulting in formulas $\langle a\rangle\varphi$ where a is one of finitely many possible *actions*. The multi-modal μ-calculus is interpreted over multi-modal Kripke structures, i.e. a structure $\mathcal{K} = \langle S, \{\xrightarrow{a} \mid a \in I\}, L\rangle$ where I is a finite set of actions and $\xrightarrow{a}$ is a transition relation for each $a \in I$. And we have $\mathcal{K}, s \models \langle a\rangle\varphi$ iff there is a successor state $s \xrightarrow{a} t$ with $\mathcal{K}, t \models \varphi$.
For the remainder of this section, let us fix a formula $\varphi \in \text{H}_\mu$ that uses k variables and a Kripke structure $\mathcal{K} = \langle S, \rightarrow, L\rangle$.
The idea of the reduction is quite simple. We use a separate copy of $\mathcal{K}$ for each possible variable assignment and additional transition relations over the

actions $\downarrow x$ and $@_x$ for a variable x that enable us to change copies or jump on the same copy. A hybrid formula $\downarrow x.\varphi$ can then simply be simulated by $\langle\downarrow x\rangle\varphi$ and analogously for jumps. To formalise this reduction we introduce two translations.
Let $\widehat{\bullet} : \mathrm{H}_\mu \to \mathrm{L}_\mu$ be the homomorphism such that

$$\begin{aligned}\widehat{\Diamond\psi} &= \langle\bullet\rangle\widehat{\psi},\\ \widehat{@_x\,\psi} &= \langle @_x\rangle\widehat{\psi},\\ \widehat{\downarrow x.\psi} &= \langle\downarrow x\rangle\widehat{\psi}\end{aligned}$$

and everything else stays the same.
Let $\widehat{\mathcal{K}} := \langle S \times (\mathit{Var} \to S), \{\xrightarrow{a} \mid a \in I\}, \widehat{L}\rangle$ be the multi-modal Kripke structure over the set of actions $I = \{\bullet\} \cup \{@_x \mid x \in \mathit{Var}\} \cup \{\downarrow x \mid x \in \mathit{Var}\}$ with $\widehat{L}(p) := \{(s,\sigma) \mid s \in L(p)\}$ for every $p \in \mathit{Prop} \cup \mathit{Nom}$ and $\widehat{L}(x) = \{(s,\sigma) \mid s = \sigma(x)\}$ for every $x \in \mathit{Var}$. The transition relations are defined through

- $(s,\sigma) \xrightarrow{\bullet} (t,\sigma)$ if and only if $s \to t$ in $\mathcal{K}$,
- $(s,\sigma) \xrightarrow{@_x} (\sigma(x),\sigma)$ for every $x \in \mathit{Var}$ and
- $(s,\sigma) \xrightarrow{\downarrow x} (s,\sigma[x \mapsto s])$ for every $x \in \mathit{Var}$,

for every $s \in S$ and $\sigma : \mathit{Var} \to S$. The following lemma reduces H_μ model checking to L_μ model checking.

Lemma 6.61. For all Kripke structures $\mathcal{K} = \langle S, \to, L\rangle$, $s \in S$, $\sigma : \mathit{Var} \to S$ and for all formulas $\varphi \in \mathrm{H}_\mu$ we have that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\widehat{\mathcal{K}}, (s,\sigma) \models \widehat{\varphi}$.

Proof. The proof is by a straightforward induction on φ. However, we need to strengthen the induction hypothesis to deal with free second-order variables. Note that the interpretation of second-order variables in H_μ is a function $\rho : \mathit{Var}_2 \to 2^{S\times(\mathit{Var}\to S)}$ while the interpretation of the second-order variables in L_μ is a only a function of type $\widehat{\rho} : \mathit{Var}_2 \to 2^{\widehat{S}}$. However, for our reduction $\widehat{S} = S \times (\mathit{Var} \to S)$ and thus ρ and $\widehat{\rho}$ are essentially the same function with the only difference being that ρ points to a tuple (s,σ) consisting of a state s and a variable assignment σ and $\widehat{\rho}$ points to a state named (s,σ). By abuse of notation we simply collapse both functions into one and prove the following statement:

For all Kripke structures $\mathcal{K} = \langle S, \to, L\rangle$, $s \in S$, $\sigma : \mathit{Var} \to S$, $\varphi \in \mathrm{H}_\mu$ and all assignments $\rho : \mathit{Var}_2 \to 2^{S\times(\mathit{Var}\to S)}$ we have that $\mathcal{K}, s, \sigma, \rho \models \varphi$ if and only if $\widehat{\mathcal{K}}, (s, \sigma), \rho \models \widehat{\varphi}$.
The statement of the lemma then simply follows for closed formulas $\varphi \in \mathrm{H}_\mu$.
Suppose first that $\varphi = p$ for $p \in \mathit{Prop} \cup \mathit{Nom}$. Then by construction of $\widehat{\mathcal{K}}$ we have that $\mathcal{K}, s, \sigma, \rho \models p$ if and only if $\widehat{\mathcal{K}}, (s, \sigma), \rho \models p$.
Now suppose that $\varphi = x$ for some $x \in \mathit{Var}$. Then $\mathcal{K}, s, \sigma, \rho \models x$ if and only if $s = \sigma(x)$. By construction of $\widehat{L}$ we thus have $(s, \sigma) \in \widehat{L}(x)$ and thus $\widehat{\mathcal{K}}, (s, \sigma), \rho \models x$.
Suppose that $\varphi = X$ for $X \in \mathit{Var}_2$. This case follows immediately with the above comment about the interpretation of second-order variables.
Negation and disjunction also follow immediately.
So, suppose that $\varphi = \Diamond\psi$. We have that $\mathcal{K}, s, \sigma, \rho \models \Diamond\psi$ if and only if there is a successor $s \to t$ with $\mathcal{K}, t, \sigma, \rho \models \psi$. By assumption we now have $\widehat{\mathcal{K}}, (t, \sigma), \rho \models \widehat{\psi}$ and by construction of $\xrightarrow{\bullet}$ we have that $(s, \sigma) \xrightarrow{\bullet} (t, \sigma)$. Thus, we also have $\widehat{\mathcal{K}}, (s, \sigma), \rho \models \langle\bullet\rangle\widehat{\psi}$, i.e. $\widehat{\mathcal{K}}, (s, \sigma), \rho \models \widehat{\varphi}$. The reverse direction can be shown similarly.
Now suppose that $\varphi = \downarrow x.\psi$. We have $\mathcal{K}, s, \sigma, \rho \models \downarrow x.\psi$ if and only if $\mathcal{K}, s, \sigma[x \mapsto s], \rho \models \psi$ and thus by the induction hypothesis $\widehat{\mathcal{K}}, (s, \sigma[x \mapsto s]), \rho \models \widehat{\psi}$. We also have that $(s, \sigma) \xrightarrow{\downarrow x} (s, \sigma[x \mapsto s])$ by construction of $\widehat{\mathcal{K}}$. Thus, we also have $\widehat{\mathcal{K}}, (s, \sigma), \rho \models \langle\downarrow x\rangle\widehat{\psi}$. The reverse direction and the case for $\varphi = @_x\, \psi$ follow in the same way.
Lastly, suppose that $\varphi = \mu X.\psi(X)$. We can show by a separate induction on α that all approximations $\mu X^\alpha.\psi(X)$ and $\mu X^\alpha.\widehat{\psi}(X)$ agree (again by slight abuse of notation in the same way as for ρ). The statement for the fixed point then follows immediately. □

Lemma 6.61 only realises a polynomial reduction for a fixed number of variables. In the general case this reduction is exponential and up to now it is not known that L_μ model checking is in P. However, it is known that L_μ model checking is in UP $\cap$ Co-UP [48], a subclass of NP $\cap$ Co-NP. Thus, at first glance we only achieve a NExpTime $\cap$ Co-NExpTime upper bound which is not optimal. However, with a slightly refined analysis we can indeed achieve an ExpTime upper bound for H_μ model checking.

Theorem 6.62. The model checking problem for H_μ is in ExpTime.

Proof. It is known that L_μ model checking for a Kripke structure $\mathcal{K}'$ and a formula $\psi \in \mathrm{L}_\mu$ can be done in time $\mathcal{O}((|\mathcal{K}'| \cdot |\psi|)^{\mathsf{ad}(\psi)})$ [32] where $\mathsf{ad}(\psi)$ denotes the fixed point alternation depth of ψ.

Now take a Kripke structure $\mathcal{K}$ and a formula $\varphi \in \mathrm{H}_\mu$ with k variables. It is not hard to see that $|\widehat{\mathcal{K}}| = \mathcal{O}(|\mathcal{K}|^{k+1})$, $|\widehat{\varphi}| = \mathcal{O}(|\varphi|)$ and $\mathsf{ad}(\widehat{\varphi}) = \mathsf{ad}(\varphi)$. Thus, model checking H_μ can be performed in time $\mathcal{O}(|\mathcal{K}|^{k+1} \cdot |\varphi|^{\mathsf{ad}(\varphi)})$, i.e. in exponential time. $\square$

On a side note we can also prove a similar result for the model checking games.

Lemma 6.63. Player $\boldsymbol{V}$ wins a position $s, \sigma \vdash \varphi$ in $\mathcal{G}(\mathcal{K}, \varphi)$ if and only if Player $\boldsymbol{V}$ wins a position $(s, \sigma) \vdash \widehat{\varphi}$ in $\widehat{\mathcal{G}}(\widehat{K}, \widehat{\varphi})$.

Proof. "$\Leftarrow$" Suppose $\boldsymbol{V}$ has a winning strategy at $(s, \sigma) \vdash \widehat{\varphi}$ in $\widehat{\mathcal{G}}(\widehat{K}, \widehat{\varphi})$. Because of positional determinacy for L_μ model checking games [85], we can assume that χ offers a choice to $\boldsymbol{V}$ at every configuration containing a disjunction or $\diamond$-formula. These choices can easily be transferred to a positional strategy χ' in $\mathcal{G}(\mathcal{K}, \varphi)$ via

$$\chi'((s,\sigma) \vdash \psi_1 \vee \psi_2) = \begin{cases} (s,\sigma) \vdash \psi_1, & \text{if } \chi(s,\sigma \vdash \widehat{\psi_1} \vee \widehat{\psi_2}) = s,\sigma \vdash \widehat{\psi_1} \\ (s,\sigma) \vdash \psi_2, & \text{otherwise} \end{cases}$$

and

$$\chi'((s,\sigma) \vdash \Diamond\psi) = (t,\sigma) \vdash \psi, \qquad \text{if } \chi(s,\sigma \vdash \langle \bullet \rangle \widehat{\psi}) = t, \sigma \vdash \widehat{\psi}.$$

Note that χ has nominally more choices in $\widehat{\mathcal{G}}(\widehat{K}, \widehat{\varphi})$ than χ' in $\mathcal{G}(\mathcal{K}, \varphi)$, especially choices involving $\langle {\downarrow} x \rangle \psi$ and $\langle @\, x \rangle \psi$ formulas. However, the underlying structure of $\widehat{K}$ is deterministic at this point, which means that $\boldsymbol{V}$'s only choice is to follow the transition relation in $\widehat{\mathcal{K}}$ which is modelled to mimic the behaviour of ${\downarrow} x.\psi$ and $@_x\, \psi$. Thus, it is not hard to see that χ' is winning if χ is winning because both games have essentially the same structure.
"$\Rightarrow$" This can be proven in the same way by transforming a winning strategy $\mathcal{G}(\mathcal{K}, \varphi)$ into one in $\widehat{\mathcal{G}}(\widehat{K}, \widehat{\varphi})$. $\square$

Putting together Lemma 6.61, Lemma 6.63 and Proposition 2.52 we also obtain Theorem 3.20 stating that the model checking games of H_μ characterise the model checking problem of H_μ for free.
For a fixed formula the model checking procedure above runs in polynomial time since the formula, as well as its alternation depth and the number of variables are fixed. We have already proven a polynomial time lower bound in Theorem 6.18. Thus, we get the following for the data complexity of H_μ.

Theorem 6.64. The data complexity for H_μ is P-complete.

By Theorem 6.17 we also get that already the expression complexity of H_μ is ExpTime-hard and thus putting this together with the upper bound from Theorem 6.62 we obtain the following result:

Theorem 6.65. Even the expression complexity of H_μ is ExpTime-complete.

6.3 Bounded Variable Fragments

As we have seen in the previous sections, for many hybrid logics the model checking complexity rises compared to their non-hybrid counterparts. This section aims to give a more refined explanation on the source of this rise in complexity. To be precise, we will study the effect of bounding the number of variables that can be used in a formula and we will see that in most cases the unbounded number of variables is the single source of these higher computational complexities.

$\mathbf{H}^k$CTL. We start again on the lowest level with HCTL and work our way up the hierarchy.

Theorem 6.66. Model Checking for H^kCTL is P-complete for every $k \in \mathbb{N}$.

Proof. The lower bound simply transfers from non-hybrid CTL model checking.
For the upper bound, consider Algorithm 2. It realises a global model checking procedure for HCTL that extends well-known bottom-up procedures for CTL.
For every subformula ψ it computes the set of states *and* variable assignments that satisfy ψ. Note that for a bounded number of variables k the number of variable assignments is bounded by $|\mathcal{K}|^k$.
Correctness and completeness can be proven by a straightforward induction on φ. The cases not involving hybrid operators are essentially the same as for CTL while the hybrid cases are similar to the corresponding cases in Algorithm 1, the decomposition algorithm used for $\mathrm{HCTL}^*_{\mathsf{ss}}$.
The algorithm – at least for fixed k – runs in polynomial time if it is implemented using dynamic programming, i.e. not re-computing subformulas that occur more than once. In this case the algorithm traverses each subformula exactly once and each clause can be computed in time $\mathcal{O}(m \cdot n \cdot n^k)$ where m is the number of edges and n is the number of nodes in $\mathcal{K}$. □

Thus, for a fixed number of variables we are only polynomially worse than normal CTL model checking. We will show in the next paragraph that the same is true for $\mathrm{H}^k\mathrm{CTL}^+$ and $\mathrm{H}^k\mathrm{FCTL}^+$.

Algorithm 2 Model checking H^kCTL.

```
procedure MC($\mathcal{K}, \varphi$)                                   $\triangleright\ \mathcal{K} = \langle S, \to, L\rangle$
    case $\varphi$ of
    $p$:  $\{(s,\sigma) \in S \times \mathit{Var} \to S \mid s \in L(p)\}$
    $x$:  $\{(s,\sigma) \in S \times \mathit{Var} \to S \mid s = \sigma(x)\}$
    $\neg\varphi'$:  $S \times \mathit{Var} \to S \setminus \mathrm{MC}(\mathcal{K}, \varphi')$
    $\varphi' \vee \varphi''$:  $\mathrm{MC}(\mathcal{K}, \varphi') \cup \mathrm{MC}(\mathcal{K}, \varphi'')$
    $\downarrow x.\varphi'$:  $\{(s,\sigma) \mid (s, \sigma[x \mapsto s]) \in \mathrm{MC}(\mathcal{K}, \varphi')\}$
    $@_x\, \varphi'$:  $\{(s,\sigma) \mid (\sigma(x), \sigma) \in \mathrm{MC}(\mathcal{K}, \varphi')\}$
    $\mathsf{EX}\varphi'$:  $\{(s,\sigma) \in S \mid \exists t \in S : (t,\sigma) \in \mathrm{MC}(\mathcal{K}, \varphi') \wedge s \to t\}$
    $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$:  $\mathrm{MC}_{\mathsf{EU}}(\mathrm{MC}(\mathcal{K}, \varphi_1), \mathrm{MC}(\mathcal{K}, \varphi_2))$
    end case
end procedure

procedure MC$_{\mathsf{EU}}$($T_1, T_2$)
$T := T_2, T' := \emptyset$
    while $T' \neq T$ do
        $T' := T$
        $T := T \cup (T_1 \cap \{(s,\sigma) \in S \times \mathit{Var} \to S \mid \exists t \in S : (t,\sigma) \in T \wedge s \to t\}$
    end while
    return $T$
end procedure
```

H^kCTL$^+$ and H^kFCTL$^+$. Remember that CTL$^+$ and FCTL$^+$ model checking is complete for $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$ [67].

Theorem 6.67. H^kCTL$^+$ and H^kFCTL$^+$ model checking is complete for Δ_2^p for every $k \in \mathbb{N}$.

Proof. The lower bound again follows from the lower bound for CTL$^+$ model checking.
For the upper bound, we give a reduction to the CTL$^+$ model checking problem borrowing ideas from the reduction of H_μ model checking to L_μ model checking in Section 6.2.4.
Basically, we build several copies of the original structure, one for each variable assignment. However, since we do not want to go to a multi-modal branching-time logic, in this case we also introduce an encoding of the $\downarrow$ and @ operators with fresh atomic propositions. As we have already seen, this reduction was in general exponential but for a fixed number of variables polynomial only.
Let us fix a Kripke structure $\mathcal{K} = \langle S, \to, L\rangle$ and a formula $\varphi \in \mathrm{H}^k$FCTL$^+$. We

will construct a structure $\widehat{\mathcal{K}}$ and a formula $\widehat{\varphi} \in \mathrm{FCTL}^+$ such that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\widehat{\mathcal{K}}, (s, \sigma) \models \widehat{\varphi}$.
Formally, $\widehat{\mathcal{K}} := \langle (S \times (\mathit{Var} \to S)) \cup (S \times (\mathit{Var} \to S) \times \{@_x, {\downarrow} x \mid x \in \mathit{Var}\}), \to, \widehat{L} \rangle$ where the new transition relation is given as follows:

- $(s, \sigma) \to (t, \sigma)$ for every $\sigma \in \mathit{Var} \to S$ iff $s \to t$ in $\mathcal{K}$.
- $(s, \sigma) \to (s, \sigma, {\downarrow} x)$ and $(s, \sigma, {\downarrow} x) \to (s, \sigma[x \mapsto s])$ for every tuple (s, σ) and $x \in \mathit{Var}$.
- $(s, \sigma) \to (s, \sigma, @_x)$ and $(s, \sigma, @_x) \to (\sigma(x), \sigma)$ for every (s, σ) and $x \in \mathit{Var}$.

For the labeling we have $\widehat{L}(p) := \{(s, \sigma) \mid s \in L(p), \sigma \in \mathit{Var} \to S\}$ for all propositions p occurring in $\mathcal{K}$ and $L(x) := \{(s, \sigma) \mid s = \sigma(x)\}$ for all $x \in \mathit{Var}$. For the new propositions we have $\widehat{L}(p_{\downarrow,x}) = \{(s, \sigma, {\downarrow} x) \mid s \in S, \sigma \in \mathit{Var} \to S\}$ as well as $\widehat{L}(p_{@,x}) = \{(s, \sigma, @_x) \mid s \in S, \sigma \in \mathit{Var} \to S\}$.
Thus, as in the case of H_μ, we have an additional copy of $\mathcal{K}$ for each possible variable assignment. The labeling on these copies is simply inherited from the labeling of the original structure and the variable assignment that it represents. Additionally, we introduce intermediary states $(s, \sigma, {\downarrow} x)$, resp. $(s, \sigma, @_x)$ labelled with fresh propositions $p_{\downarrow,x}$ or $p_{@,x}$ to model the respective hybrid operators. For example, ${\downarrow} x.\varphi$ at a state s with regard to a variable assignment σ can be performed by finding the unique path $(s, \sigma) \to (s, \sigma, {\downarrow} x) \to (s, \sigma[x \mapsto s])$ in $\widehat{\mathcal{K}}$. Thus, the formula can be changed to find the (unique) successor satisfying $p_{\downarrow,x}$. A similar pattern may be performed to simulate jumps.
Note that the structure $\widehat{\mathcal{K}}$ has $(k+1) \cdot |\mathcal{K}|^{(k+1)}$ many states and thus is only polynomially larger than $\mathcal{K}$ for fixed k.
The structure $\widehat{\mathcal{K}}$ now contains paths that are used to change copies or to jump back to some state. The formula needs to deal with this and also needs to simulate the ${\downarrow} x$ and $@_x$ operators.
Formally, let $\varphi \mapsto \widehat{\varphi}$ be the homomorphism such that

$$\begin{aligned}
\widehat{{\downarrow} x.\varphi} &:= \mathsf{EX}(p_{\downarrow,x} \wedge \mathsf{EX}\varphi), \\
\widehat{@_x \varphi} &:= \mathsf{EX}(p_{@,x} \wedge \mathsf{EX}\varphi) \text{ and} \\
\widehat{\mathsf{E}\psi} &:= \mathsf{E}(\mathsf{G}(\bigwedge_{x \in \mathit{Var}} \neg p_{\downarrow,x} \wedge \neg p_{@,x}) \wedge \widehat{\psi}).
\end{aligned}$$

It is easy to see that $\widehat{\varphi}$ is only polynomially larger than φ. A straightforward induction combined with the ideas above proves that $\mathcal{K}, s, \sigma \models \varphi$ if and only if $\widehat{\mathcal{K}}, (s, \sigma) \models \widehat{\varphi}$.

This reduction combined with the Δ_2^p upper bound for FCTL$^+$ model checking also proves a Δ_2^p upper bound for H^{k}FCTL$^+$ model checking.
The same reduction also holds for H^{k}CTL$^+$. Note for this that the reduction on the formula level does not introduce any GF-operators. $\square$

H^{k}CTL*. For HCTL$^*_{\mathsf{ss}}$ the complexity compared to CTL* model checking did not rise and it is also known that already CTL* model checking is PSpace-complete. Thus, we obtain the same for the bounded fragment.

Theorem 6.68. H^{k}CTL$^*_{\mathsf{ss}}$ model checking is PSpace-complete for every $k \in \mathbb{N}$.

Interestingly, however, the complexity for H^{k}CTL$^*_{\mathsf{ps}}$ model checking also drops to PSpace. The lower bound follows directly from the previous result since H^{k}CTL$^*_{\mathsf{ps}}$ is an extension of H^{k}CTL$^*_{\mathsf{ss}}$. The upper bound can be proven similarly to Theorem 6.38. Notice that, for a bounded number of variables, the size of the model checking games is only exponential and a similar decomposition algorithm then leads to a PSpace procedure.

Theorem 6.69. H^{k}CTL$^*_{\mathsf{ps}}$ model checking is PSpace-complete for every $k \in \mathbb{N}$.

Lastly, we consider H^{k}CTL$^*_{\mathsf{pp}}$. Here, we need to distinguish the special case with only one variable which we will show to have an elementary model checking procedure while the cases with at least two variables are as hard as the cases with an unbounded number of variables.

Theorem 6.70. H^{k}CTL$^*_{\mathsf{pp}}$ model checking for $k \geq 2$ is m-ExpSpace hard for every $m \geq 1$ and ExpSpace-hard for $k = 1$.

Proof. For this, observe that the formula $\varphi_{\mathcal{T},n,m}$ that encodes the 2^n_m-corridor tiling problem developed in Section 6.1.3 on level $m = 1$ only uses one variable while the cases for $m > 1$ use two variables. Thus, we can deduce ExpSpace-hardness for the one-variable fragment and m-ExpSpace hardness for every m for the fragments using at least two variables. $\square$

Thus, the cases using at least two variables are at least as difficult as the unbounded case and we do not need to prove another upper bound. For the case using only one variable we show a matching upper bound.

Theorem 6.71. H^{1}CTL$^*_{\mathsf{pp}}$ model checking is in ExpSpace.

	Model Checking Complexity				
	combined	expression	data	bounded fragments	
				$k = 1$	$k \geq 2$
H_μ	ExpTime	ExpTime	P	NP $\cap$ Co-NP P	
$\mathrm{HCTL}^*_{\mathsf{pp}}$	Tower Elementary	Tower Elementary	PSpace	ExpSpace	Tower Elementary
$\mathrm{HCTL}^*_{\mathsf{ps}}$	ExpSpace	ExpSpace	PSpace	PSpace	
$\mathrm{HCTL}^*_{\mathsf{ss}}$	PSpace	PSpace	NLogSpace	PSpace	
HFCTL^+	PSpace	PSpace	NLogSpace	Δ^p_2	
HCTL^+	PSpace	PSpace	NLogSpace	Δ^p_2	
HCTL	PSpace	PSpace	NLogSpace	P	

Figure 6.7: The complexities of model checking hybrid branching-time logics.

Proof. We use the same decision procedure as described in Theorem 6.54. The crucial step that accounts for the complexity is constructing the NBA for a formula of type $\mathsf{E}\psi$. Notice that $\mathsf{E}\psi \in \mathrm{H}^1\mathrm{CTL}^*_{\mathsf{pp}}$ and thus, ψ is a hybrid LTL formula with only one variable. Instead of constructing a Büchi-automaton, we proceed as in [83, Prop 4.5] and build an alternating one-pebble Büchi automaton that acccepts all the paths satisfying ψ and which is polynomial in $|\psi|$. With [83, Thm 3.1] we can then translate this pebble-automaton into an NBA of size doubly exponential in the size of ψ. The theorem then follows with Lemma 6.48. □

6.4 A Complete Picture

The summary of all results about model checking hybrid branching-time logics is depicted in Figure 6.7. As a comparison, Figure 6.8 lists the model checking complexities of the classical branching-time logics. A single entry means that the problem is complete for the mentioned complexity class. A multicolumn entry depicts lower and upper bounds.

All model checking problems of hybrid branching-time logics – except those for $\mathrm{HCTL}^*_{\mathsf{pp}}$ and the bounded fragments of H_μ– are complete for their respective complexity classes and thus the decision procedures presented in this chapter are optimal. In the case of $\mathrm{HCTL}^*_{\mathsf{pp}}$ note that Elementary does not have complete problems.

Model Checking	Complexity
L_μ	NP ∩ Co-NP P
CTL*	PSpace
FCTL$^+$	Δ_2^p
CTL$^+$	Δ_2^p
CTL	P

Figure 6.8: The complexities of model checking branching-time logics.

Generally, for the hybrid logics below HCTL$^*_{\mathsf{ss}}$, the complexity of their model checking problems has increased compared to their non-hybrid counterparts. Their model checking problem is now in all cases PSpace-complete. Surprisingly then, HCTL$^*_{\mathsf{ss}}$ has also only a PSpace-complete model checking problem which means that from a complexity theoretic point of view it is not harder to model check HCTL$^*_{\mathsf{ss}}$ formulas than ordinary CTL* formulas despite its vast increase in expressive power between these two logics.

However, in the branching-time hierarchy this is the only case where model checking is not harder than its non-hybrid counterparts. Going further up in the hierarchy to HCTL$^*_{\mathsf{ps}}$ and HCTL$^*_{\mathsf{pp}}$ we have that the computational cost of model checking then rises dramatically first to ExpSpace and then to levels that do not admit any elementary bound on the model checking algorithms. Considering their increased expressive power however this is not too surprising.

Finally, the fully hybrid μ-calculus has an ExpTime-complete model checking problem. On the one hand this is a clear theoretical increase in complexity for the model checking problem compared to L_μ. But on the other hand, most algorithms for L_μ model checking only run in exponential time anyways [39, 90]. Thus, from a practical point of view this increase may not be too bad.

We have also looked at the data and expression complexities of these model checking problems. Quite surprisingly, the expression complexity of all logics is as complex as the combined complexity, showing again that these hybrid logics are quite expressive even on fixed structures. The data complexity, which is more important for practical purposes, is significantly lower. The logics from HCTL to HCTL$^*_{\mathsf{ss}}$ are only NLogSpace-complete with respect to a fixed formula. This suggests that the presented model checking algorithms

may even prove to be useful for practical purposes in which formulas are typically fairly small compared to the size of the system. The data complexity of H_μ is P-complete and thus also quite low. Only $\mathrm{HCTL}^*_{\mathsf{pp}}$ and $\mathrm{HCTL}^*_{\mathsf{ps}}$ are worse with a PSPACE-complete data complexity.
Another indicator that these algorithms may perform reasonably well in practice can be seen if we look at the bounded fragments of these logics. Here, we can see that most of the increases in complexity originate from the unbounded use of variables.
Bounded fragments, in which the number of variables are bounded, have generally – with the exception of $\mathrm{HCTL}^*_{\mathsf{pp}}$– the same computational complexity as their non-hybrid counterparts. The increase in complexity for $\mathrm{H}^k\mathrm{CTL}^*_{\mathsf{pp}}$ on the other hand comes from the ability to also store the exact moment on a path and to reference this moment with jumps which – already with two variables – enables them to express a limited form of recursion (in terms of the temporal depth of a formula) on a path between two points. This is no longer possible with only one variable and thus $\mathrm{H}^1\mathrm{CTL}^*_{\mathsf{pp}}$ is only EXPSPACE-comlete with regard to model checking.

Chapter 7

Satisfiability and Decidable Fragments

Satisfiability for hybrid logics – especially in the presence of the binder – is almost always undecidable, even on the level of hybrid modal logic [5, 7, 69]. Other hybrid concepts like nominals or jumps do not pose such difficult problems and mostly remain decidable.

This naturally raises the question of decidable fragments. There are usually two courses of action in regaining decidability. Either one poses syntactic restrictions on the used operators to possibly obtain an "easier" logic or one can limit the structures on which the logic is interpreted on. The first is usually referred to as a syntactic restriction while the second approach that imposes restrictions on the semantical side is referred to as a semantic restriction.

In this chapter we will pursue both types of restrictions. After a short summary of the undecidability proofs for our hybrid logics that show that hybrid logics on the most basic level are already undecidable we will start our search for decidable fragments with syntactic restrictions. After this we will also look at semantic restrictions – hybrid logics on tree structures.

Generally speaking however, these fragments – while still being extensions of the non-hybrid logics – are quite limited in their expressive power compared to the full hybrid logics. This is because the hybrid framework is quite powerful so we need to make strong restrictions to regain decidability. As a consequence the techniques used to show decidability of these fragments are also quite rudimentary. So this chapter should be understood as only a first look at decidability issues for hybrid branching-time logics.

7.1 Undecidability of the Full Hybrid Logic

It is known that already hybrid modal logic with binder and jump is undecidable [5, 7, 69]. Thus, since HCTL and all other logics considered in this thesis are extensions thereof, they are also undecidable.
However, we will give a slightly modified proof for the undecidability of HCTL and its extensions which does not use the X-operator. We thus show that even HCTL without the X-operator, i.e. with only U and hybrid operators, is undecidable. We further show that we only need three variables. This means that also bounding the number of variables generally does not help much. Our proof extends previous proofs for the undecidability of hybrid modal logics and also shows that the undecidability does not originate from the interplay between the X-operator and hybrid operators.

Theorem 7.1. H^3CTL is undecidable.

Proof. We will reduce the unbounded corridor tiling problem to the Satisfiability problem of H^3CTL, i.e. given a tiling system $\mathcal{T}$, we will construct a formula $\varphi_{\mathcal{T}}$ that is satisfiable if and only if there exists a valid $\mathcal{T}$-tiling of the unbounded corridor.
For this, let $\mathcal{T} = (T, H, V)$ be a tiling system. The formula $\varphi_{\mathcal{T}}$ consists of two parts. The first part simply enforces a "grid-like" structure on the model that is unbounded to the left and to the top. As mentioned above, this proof does not use the X-operator. Instead, we define an alternative Next-operator with hybrid operators and Until only:

$$\mathsf{EX}'\varphi \;:=\; \downarrow w.\mathsf{E}(w\mathsf{U}(\neg w \wedge \varphi)),$$

stating that – aside from self-loops that may occur on this state – there is a direct successor of the current state which is not the current state. Similarly we can define $\mathsf{AX}'\varphi \;:=\; \downarrow w.\mathsf{A}(w\mathsf{U}(\neg w \wedge \varphi))$. Note, that this universally quantified formula also prohibits self-loops since all paths necessarily need to leave the current state and in the case of a self-loop there is of course one path that always loops back to the current state.
After encoding the grid, the second part of the formula then only has to verify that the tiles are placed in a valid manner on top of this grid.
As before, when encoding tiling systems we use the tiling set T as atomic propositions. Furthermore, we use four additional atomic propositions $\{p_{00}, p_{10}, p_{01}, p_{11}\}$ which help us in encoding the grid-structure.
We begin with the grid-like structure. The grid will be encoded as shown in Figure 7.1.

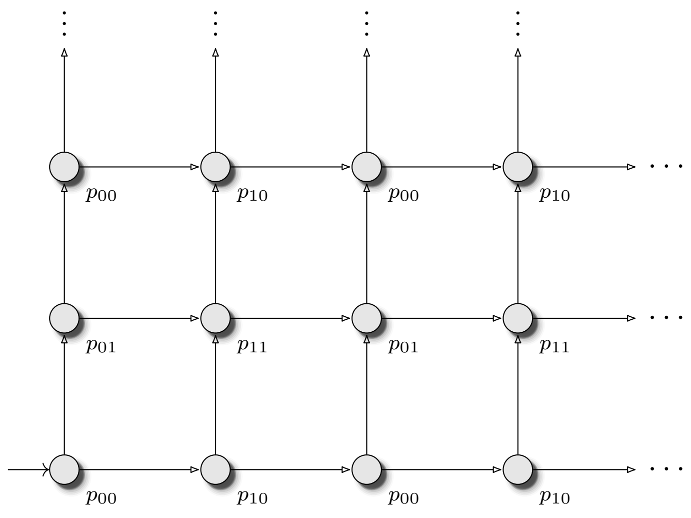


Figure 7.1: The grid as enforced by the formula φ_{grid}.

To initialise the grid, we start with p_{00} and state that there are no cycles – not even self-loops – in the structure:

$$\varphi_{\mathsf{init}} \;:=\; p_{00} \wedge \mathsf{AG}{\downarrow} x.\mathsf{AX}'\mathsf{AG}\neg x.$$

Then we need to make sure that on each state exactly one of the p_{ij} for $i, j \in \{0, 1\}$ holds:

$$\varphi_{1!} \;:=\; \mathsf{AG}\Big(\bigvee_{i,j\in\{0,1\}} p_{ij} \wedge \bigwedge_{(i',j')\neq(i,j)} \neg p_{i'j'}\Big).$$

The next formula requires each state to have exactly two successors:

$$\varphi_{2!} \;:=\; \mathsf{AG}{\downarrow} x.\mathsf{EX}'({\downarrow} y.\, @_x\, \mathsf{EX}'(\neg y \wedge {\downarrow} z.\, @_x\, \mathsf{AX}'(y \vee z))).$$

Then we require that both successors are correctly labelled as shown in Figure 7.1. A right neighbor increases the first index modulo 2 while a neighbor on top increases the second index. This is done via

$$\varphi_{\mathsf{lab}} \;:=\; \mathsf{AG}\Big(\bigwedge_{i,j\in\{0,1\}} p_{ij} \rightarrow \mathsf{EX}'p_{(i+1)j} \wedge \mathsf{EX}'p_{i(j+1)}\Big),$$

where the indices are all understood to be modulo 2. Note that this formula relies both on $\varphi_{1!}$ and $\varphi_{2!}$ and does not work independently.

Finally, we will use this labelling to enforce a grid-structure:

$$\begin{aligned}\varphi_{\mathsf{gr}} \;&:=\; \mathsf{AG}\Big(\bigwedge_{i,j\in\{0,1\}} p_{ij} \rightarrow {\downarrow} x.\mathsf{EX}'\big(p_{(i+1)j} \wedge \mathsf{EX}'\big(p_{(i+1)(j+1)} \wedge \\ &\qquad\qquad {\downarrow} y.\, @_x\, \mathsf{EX}'\big(p_{i(j+1)} \wedge \mathsf{EX}'\big(p_{(i+1)(j+1)} \wedge y\big)\big)\big)\big)\Big).\end{aligned}$$

Again, all indices are understood modulo 2. The formula states that it does not matter if we first go to the right and then to the successor on top or first to the successor on top and then to the right, on both ways we reach the same state.
We obtain φ_{grid} as the conjunction of $\varphi_{\mathsf{init}}, \varphi_{1!}, \varphi_{2!}$ and φ_{gr}. This concludes the first part of the formula.
The second part now tries to encode a valid $\mathcal{T}$-tiling on this grid. With the chosen encoding this is particularly easy since horizontal and vertical matching relations can simply be verified by inspecting both successors. Thus we get

$$\varphi_{\mathsf{tiling}} \;:=\; t_1 \wedge \mathsf{AG}\Big(\bigvee_{t\in T} t \wedge \bigwedge_{t'\neq t} \neg t'\Big) \wedge \varphi_{\mathsf{hor}} \wedge \varphi_{\mathsf{ver}}$$

with

$$\begin{aligned}\varphi_{\mathsf{hor}} \;&:=\; \mathsf{AG}\Big(\bigwedge_{i,j\in\{0,1\}} \bigwedge_{t\in T} p_{ij} \wedge t \rightarrow \mathsf{AX}'\big(p_{(i+1)j} \rightarrow \bigvee_{(t,t')\in H} t'\big)\Big) \\ \varphi_{\mathsf{ver}} \;&:=\; \mathsf{AG}\Big(\bigwedge_{i,j\in\{0,1\}} \bigwedge_{t\in T} p_{ij} \wedge t \rightarrow \mathsf{AX}'\big(p_{i(j+1)} \rightarrow \bigvee_{(t,t')\in V} t'\big)\Big).\end{aligned}$$

We obtain $\varphi_{\mathcal{T}} \;:=\; \varphi_{\mathsf{grid}} \wedge \varphi_{\mathsf{tiling}}$. With the explanations above it should be clear that $\varphi_{\mathcal{T}}$ is satisfiable if and only if $\mathcal{T}$ admits a valid $\mathcal{T}$-tiling on the unbounded corridor. And thus, since already the unbounded corridor tiling problem is undecidable with Proposition 2.68, we also obtain undecidability of H^{3}CTL and all of its extensions. $\square$

This reduction proves hardness for Π^0_1 in the arithmetical hierarchy. It is quite interesting but in the presence of fairness operators not too surprising to see that already HFCTL$^+$, which is slightly more expressive than HCTL (see Chapter 5), can be shown to be highly undecidable.

Theorem 7.2. H^{3}FCTL$^+$ is hard for Σ^1_1.

Proof. We prove this by a reduction from the recurring unbounded corridor tiling problem which is Σ^1_1-complete according to Proposition 2.69.
So, let $\mathcal{T} = (T, H, V)$ be a tiling system and $t_\infty \in T$ a special tile that is supposed to occur infinitely often in the first column. Define $\mathcal{T}' = (T \cup \{t'_\infty\}, H', V')$ with

$$\begin{aligned} H' &:= H \cup \{(t'_\infty, t) \mid (t_\infty, t) \in H\}, \\ V' &:= V \cup \{(t'_\infty, t) \mid (t_\infty, t) \in V\} \cup \{(t, t'_\infty) \mid (t, t_\infty) \in V\}. \end{aligned}$$

Thus, $\mathcal{T}'$ has a second copy of the tile that is supposed to occur infinitely often. Note that the horizontal matching relation is only extended in such a way that t'_∞ can only be placed in the first column, while the vertical matching relation is simply extended such that t'_∞ and t_∞ have the same possibilities of vertically aligned dominoes.
It is easy to see that there is a recurring $\mathcal{T}$-tiling of the unbounded corridor in which t_∞ occurs infinitely often in the first column if and only if there is a recurring $\mathcal{T}'$-tiling of the unbounded corridor in which t'_∞ occurs infinitely often in the first column.
To finish the reduction, we now take the formula $\varphi_{\mathcal{T}'}$ constructed in the previous proof but instead of t_1 at the origin we require that $\mathsf{EGF}t'_\infty$. With the previous change to the tiling system it should be clear that if there is a path on which t'_∞ occurs infinitely often then this path needs to cover the first column of the tiling, since t'_∞ can only be placed there. $\square$

7.2 Decidable Fragments

A natural question that immediately arises after seeing that already H^{3}CTL is undecidable is to find fragments of these logics (which obviously should still extend the non-hybrid logics) that are still decidable.
We begin our search with syntactic fragments, i.e. logics in which the formulas are syntactically restricted, either by disallowing certain operators or imposing conditions on how these operators can be used. A naive idea to regain decidability via syntactic restrictions on the logic would be to limit the use of binders by bounding the number of variables that can be used even further since we have only proven undecidability for 3 or more variables. However, it was proven that hybrid modal logic is already undecidable even if only one nominal and one variable is used [89]. Thus, this approach seems to not be very promising and is not pursued further.
Another approach seen in [89] to regain decidability on the level of hybrid modal logic is to restrict the interplay between the universal modal

$\Box$-operator and the binder. Intuitively, if a $\downarrow$ does not occur under any universal quantifiers like $\Box$ or G, then it only gets bound to a single state during the evaluation and thus the variable can be replaced by a nominal pointing to this unique state.
Indeed, in [89] it was even shown that the restricted logic, in which formulas containing the pattern $\Box\downarrow\Box$, i.e. a box-operator nested underneath a $\downarrow$ nested underneath a box in the syntax tree, are forbidden, is decidable and even 2-ExpTime complete. The proof is a combination of eliminating binders with the previous idea and a translation into guarded logic which is known to be decidable.
However, the translation into guarded logic seems to not be feasible in the presence of temporal operatos. The guarded fragment of first-order logic for example is not capable of expressing reachability and thus needs to be extended to subsume these hybrid branching-time logics. But by adding least fixed points which would be able to express the temporal operators we lose the guardedness property in the translation of hybrid branching-time logics. And of course first-order logic with added least fixed points is in general undecidable and thus this part of the approach does not help us in the search for decidable fragments.
In this section we will extend the first part of the approach, binder elimination of bound variables that are not under universal operators up to the level of $\mathrm{HCTL}^*_{\mathsf{pp}}$ and present decidable fragments. This also extends and is similar to an approach to find decidable fragments for HCTL by [10].
For this we first show that the fragment $\mathrm{HCTL}^*_{@}$ that disallows binders completely is decidable and then continue to present a richer fragment that allows a restricted use of the binder and reduce this fragment to $\mathrm{HCTL}^*_{@}$ via binder elimination. Similarly, the fragment of H_μ in which binder is disallowed is referred to as $\mathrm{H}_{\mu,@}$.

7.2.1 $\mathrm{H}_{\mu,@}$ and $\mathrm{HCTL}^*_{@}$

By definition $\mathrm{H}_{\mu,@}$ coincides with the hybrid μ-calculus as introduced by Sattler and Vardi in [80]. They have already shown that $\mathrm{H}_{\mu,@}$ is decidable.

Theorem 7.3 ([80])**.** $\mathrm{H}_{\mu,@}$ is decidable in ExpTime.

It should be clear that if we disallow the binder completely that the hierarchy of hybrid logics from $\mathrm{HCTL}^*_{\mathsf{ss}}$ to $\mathrm{HCTL}^*_{\mathsf{pp}}$ collapses into a single logic. The collapse from $\mathrm{HCTL}^*_{\mathsf{ps}}$ to $\mathrm{HCTL}^*_{\mathsf{ss}}$ without binders is obvious since they only differ in the use of the binder, thus their respective fragments with no binder at all coincide. Similarly, $\mathrm{HCTL}^*_{\mathsf{pp}}$ only extends $\mathrm{HCTL}^*_{\mathsf{ps}}$ by the use

of jumps underneath a binder on a path formula. Thus, disallowing binders also disallows these jumps.
We simply refer to the fragment of $\text{HCTL}^*_{\mathsf{ss}}/\text{HCTL}^*_{\mathsf{ps}}/\text{HCTL}^*_{\mathsf{pp}}$ that disallows binders as $\text{HCTL}^*_{@}$. We can now make use of the connection between $\text{HCTL}^*_{\mathsf{ss}}$ and H_μ that was presented in Section 5.2.2 to obtain a simple decision procedure for $\text{HCTL}^*_{@}$.

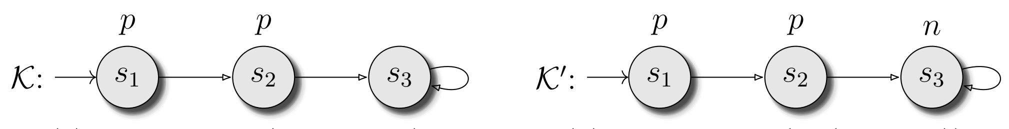
Theorem 7.4. $\text{HCTL}^*_{@}$ is decidable in 3-ExpTime.

Proof. By Theorem 5.28 we can translate any $\text{HCTL}^*_{\mathsf{ss}}$ formula into an equivalent H_μ formula. The blowup involved is doubly-exponential due to the Büchi-automata involved in translating path formulas. A closer inspection also shows that during the translation no additional binders are introduced. Thus, since $\text{HCTL}^*_{@}$ is a fragment of $\text{HCTL}^*_{\mathsf{ss}}$, we can use the same procedure to translate an $\text{HCTL}^*_{@}$ formula into an equivalent $\text{H}_{\mu,@}$ formula that is exponentially larger.
By Theorem 7.3, $\text{H}_{\mu,@}$ is decidable in ExpTime. Putting both observations together gives us a 3-ExpTime procedure for satisfiability of $\text{HCTL}^*_{@}$. □

CTL* has a 2-ExpTime-complete satisfiability problem [92] so we get this lower bound from the fact that $\text{CTL}^* \subseteq \text{HCTL}^*_{@}$. However the exact complexity of satisfiability for $\text{HCTL}^*_{@}$ remains unknown for now.
With this, yet again, we observe that the binder is the most critical operator when considering the satisfiability problem. Naturally, one can now ask if there is a restricted way to add the binder such that the logic remains decidable. One – still quite restrictive – way is shown in the following section. As previously mentioned this approach generalises a technique by [10] and the approach of [89].

7.2.2 The Reducible/Existential Fragment of HCTL*

Take a look at the formula $\mathsf{E}(p\mathsf{U}\downarrow x.\mathsf{EX}x)$ which states that there is a path on which p holds up to a point that has a self-loop. Clearly this formula is satisfiable. For example Figure 7.2a shows a model that satisfies the formula at s_1 where x can be bound to the state s_3. But more importantly, we can observe that the binder in this formula is *existential* in its nature, i.e. it is only bound once during the evaluation of this formula. This is because it only occurs underneath the right-hand part of an Until-formula which is satisfied in a single state only.
The idea now is that a variable which – for syntactic reasons only – gets bound at most once can also be treated as a simple nominal. For example it is quite obvious that the formula $\mathsf{E}(p\mathsf{U}(n \wedge \mathsf{EX}n))$ is equi-satisfiable to the



(a) A model of $\mathsf{E}(p\mathsf{U}\downarrow x.\mathsf{EX}x)$. (b) A model of $\mathsf{E}(p\mathsf{U}(n \wedge \mathsf{EX}n))$.

Figure 7.2: Correlation of models when eliminating existential occurrences of the binder.

previous one. A model is shown in Figure 7.2b. It is also not too difficult to see that the models of both formulas share a close connection: a model of the first formula can simply be translated into a model of the second formula by naming the state on which the variable would be bound. And if a structure does not satisfy the first formula then there is also no way to place a nominal on it such that the second formula is satisfied.
However, the second formula falls into the category of $\text{HCTL}^*_{@}$ which was shown to be decidable above.
In the remainder of this section we generalise this idea. We begin by defining what it means for an occurrence of a binder in a formula to be *existential* or *universal*.
For this, remember that every $\text{HCTL}^*_{\mathsf{pp}}$ formula admits a negation normal form in which negation only occurs in front of atomic formulas but which features not only X and U as native operators, but also G. The latter operator is introduced when pushing negation inside the Until operator, for example $\neg(p\mathsf{U}q) \equiv (\mathsf{G}\neg q) \vee (\neg q\mathsf{U}(\neg p \wedge \neg q))$.

Definition 7.5. Let $\varphi \in \text{HCTL}^*_{\mathsf{pp}}$ in negation normal form. We say that an occurrence $\downarrow x.\psi \in \mathsf{Sub}(\varphi)$ is *universal* with respect to φ if it occurs underneath an A path-quantifier, underneath a G-operator or underneath the left-hand side of an U-operator in the syntax tree of φ. Otherwise, this occurrence is called *existential.*
A formula $\varphi \in \text{HCTL}^*_{\mathsf{pp}}$ is called *existential* if all subformulas of the form $\downarrow x.\chi \in \mathsf{Sub}(\varphi)$ are existential. Otherwise the formula is *universal.*
The *existential fragment of* $\text{HCTL}^*_{\mathsf{pp}}$ is the set of all $\text{HCTL}^*_{\mathsf{pp}}$ formulas whose negation normal form is existential.

The existential fragment of $\text{HCTL}^*_{\mathsf{pp}}$ is a superset of the reducible fragment of HCTL introduced in [10]. Similar to the reducible fragment it is not closed under negation.

Example 7.6. The formula $\mathsf{E}(p\mathsf{U}\downarrow x.\mathsf{EX}x)$ belongs to the existential fragment, since the binder only occurs under an existential path quantifier and is on the right-hand side of the Until in the formula. In fact, it already belongs to the reducible fragment of HCTL. However, the formula $\neg\mathsf{E}(p\mathsf{U}{\downarrow}x.\mathsf{EX}x) \equiv \mathsf{A}((\mathsf{G}{\downarrow}x.\mathsf{AX}\neg x) \vee (({\downarrow}x.\mathsf{AX}\neg x)\mathsf{U}\neg p))$ does not belong to the existential fragment.

Theorem 7.7. Satisfiability for the existential fragment of $\mathrm{HCTL}^*_{\mathsf{pp}}$ is decidable in 3-ExpTime.

Proof. First, we describe a polynomial procedure that eliminates all existential occurrences of the binder in an $\mathrm{HCTL}^*_{\mathsf{pp}}$ formula while preserving satisfiability.
For this, let φ be a formula in the existential fragment. W.l.o.g. we assume that φ is in negation normal form and we assume that each variable in φ gets bound at most once and these variables are named $x_1, \dots, x_k$ for some $k \in \mathbb{N}$. Then

$$\varphi' \; := \; \varphi\Big[(n_1 \wedge \varphi_1(n_1))/({\downarrow}x_1.\varphi_1(x_1)), \dots (n_k \wedge \varphi_k(n_k))/({\downarrow}x_k.\varphi_k(x_k))\Big],$$

where $n_1, \dots, n_k$ are fresh nominals that did not occur in φ, and $\varphi_i(n_i)$ indicates that every free occurrence of x_i in φ_i is replaced by n_i.
One can prove by a straightforward induction on φ that φ is satisfiable if and only if φ' is satisfiable. The key ingredient is that each subformula ${\downarrow}x_i.\varphi_i$ that gets replaced can only occur underneath E-path quantifiers, X-operators and underneath the right-hand side of U. Thus, if φ is satisfied then x_i gets bound to a single state during the evaluation of φ. Thus, models of φ can simply be translated to a model of φ' by picking this unique state for the newly introduced nominals. The reverse direction holds as well. For more details, cf. [10, Thm 4.3] which can easily be adapted to our case.
Finally, observe that $\varphi' \in \mathrm{HCTL}^*_{@}$ which is decidable in 3-ExpTime by Theorem 7.4. □

7.2.3 Hybrid Branching-Time Logic on Trees

So as we have seen, on the side of syntactic restrictions we can regain decidability by restraining the use of the binder heavily.
On the other side, it is often also interesting to only look at certain classes of Kripke structures. Such restrictions are often referred to as semantic restrictions of the logic. As we have already seen in Chapter 5 the expressive power of hybrid logic can vary depending on the class of structures in question. One very important class of structures are computation trees which are often used to model the possible step-by-step behaviour of reactive systems.

With the previous work on $\text{HCTL}^*_{\mathsf{pp}}$ on trees in Section 5.3 it is easy to see that for hybrid branching-time logics interpreted only on trees decidability is immediately regained.

Theorem 7.8. $\text{HCTL}^*_{\mathsf{pp}}$ on trees is deciable in Tower.

Proof. We have already seen in Theorem 5.42 that $\text{HCTL}^*_{\mathsf{pp}}$ on trees can be translated into MPL which in turn is a fragment of MSO.
It is also known that MSO on trees is decidable (cf. [47]). This result is obtained by a translation to parity tree automata for which emptiness can easily be checked. However, negation in the formulas is handled by complementing the corresponding automaton which involves an exponential blowup. Thus, this procedure has no elementary bound. □

Because the complexity originates in negation rather than the temporal or hybrid operators this sadly does not drop if we only consider the fragments CTL, CTL^+, FCTL^+, $\text{HCTL}^*_{\mathsf{ss}}$ or $\text{HCTL}^*_{\mathsf{ps}}$.

Chapter 8

Conclusion

We conclude this thesis with some general thoughts about hybrid branching-time logics as well as some ideas for further research in this area.

General Thoughts on Hybrid Branching-Time Logics. The hybrid framework in general has proven to be a very versatile and highly expressive addition to classical branching-time logics. Many "structural" properties that cannot be expressed by standard branching-time logics are expressible through the addition of this framework. This also includes many extensions of branching-time logics proposed over the years. A connection to some of these extensions has been shown in Section 3.7.

Generally speaking, the additional expressive power provided by the hybrid operators is orthogonal to the expressive power achieved via classical branching-time features. This means that in general hybrid operators cannot be used to simulate fairness-constraints or nesting of path formulas. We have shown this in Chapter 5. The former result was already proven on trees by [52]. To show the latter result we have developed Ehrenfeucht-Fraïssé games that helped us to capture the expressive power of HCTL. We have proven that HCTL on finite structures coincides with HFCTL^+ and thus separating HCTL from $\mathrm{HCTL}^*_{\mathsf{ss}}$ on finite structures also showed that HFCTL^+ is less expressive than $\mathrm{HCTL}^*_{\mathsf{ss}}$.

We have also shown that $\mathrm{HCTL}^*_{\mathsf{ss}}$ can be translated into the hybrid μ-calculus. In doing so we have also established that the bounded fragments of $\mathrm{HCTL}^*_{\mathsf{ss}}$ are invariant under k-bisimulations. This was already shown in Chapter 4 for the bounded fragments of H_μ. And finally, since $\mathrm{HCTL}^*_{\mathsf{ps}}$ can express properties that are not invariant under k-bisimulations we have also obtained that allowing changes to the variable assignment along the evaluation of a path truly increases the expressive power.

This picture changes quite a bit if the class of structures is restricted. At least on some restricted classes of structures the hybrid framework can be used to simulate fairness constraints or nesting of path formulas. For example on finite structures we can use that for something to occur infinitely often along some path it has to occur on a (finite) cycle. Hence, even HCTL is able to express fairness properties on finite structures. And on trees we can use the fact that each state has a unique parent to simulate some nesting of path formulas. Thus we can express some exclusive CTL* properties already in HCTL$^+$ and thus also in HCTL on trees.

This additional expressive power however comes at a price. We have seen and discussed in Chapter 7 that even the most basic hybrid branching-time logic, in fact even hybrid modal logics, are undecidable as soon as the binder is added. We have discussed some possibilities to regain decidablity but the restrictions needed seem to be quite harsh and frankly not very useful. One line of research in this direction would be to consider fragments in which the binder can be completely eliminated. Our translation to the hybrid μ-calculus then provides a way to use the decidability results for the hybrid μ-calculus without binders by Sattler and Vardi. Another is to consider only the class of trees which regains decidability automatically since all hybrid branching-time logics on trees can be translated into Monadic Second-Order logic. However, we only achieve a nonelementary complexity in this case.

For model checking the picture we have obtained in Chapter 6 is not quite so bleak. The model checking problem for all hybrid branching-time logics is still decidable. However, compared to the classical branching-time logics we generally have a slight increase in computational complexity.

We have shown that already HCTL is PSpace-complete. Thus, model checking the hybrid version is considerably harder than CTL which is known to be in P. However, quite surprisingly, this increase in complexity does not continue for the next logics. All hybrid logics up to HCTL$^*_{\mathsf{ss}}$ remain in PSpace and have no further increase in complexity. Thus, HCTL$^*_{\mathsf{ss}}$ is not harder to model check than CTL* which is also known to be PSpace-complete.

Above HCTL$^*_{\mathsf{ss}}$ we see an increase in complexity again. While HCTL$^*_{\mathsf{ps}}$ is ExpSpace-complete, HCTL$^*_{\mathsf{pp}}$ has no elementary bound whatsoever. And finally H$_\mu$ is ExpTime-complete.

This general increase in complexity aside from the case for HCTL$^*_{\mathsf{ss}}$ may look worse than it actually is. Further analysis has revealed that this increase mainly comes from the unbounded number of variables. For the bounded fragments model checking is only polynomially worse than the non-hybrid logics. Thus, especially for small formulas with only few variables model checking in practice might still be somewhat viable.

Further Research. Finally, we want to discuss some open questions and possibilities for further research in the field of hybrid branching-time logics. We have presented a detailed analysis of the model checking problem for all hybrid branching-time logics. The model checking algorithms are fairly straightforward and conceptually easy extensions to the ones for classical branching-time logics. Our research also suggests that the increase in computational complexity might not be too bad in practice where the formulas tend to be quite small. An implementation of these algorithms and an evaluation of their performance would be a good next step.
The biggest open question concerning the analysis of the relative expressive power of these new hybrid branching-time logics is the logic $\text{HCTL}^*_{\mathsf{pp}}$. We have not yet been able to establish a precise connection to its fragments, especially with $\text{HCTL}^*_{\mathsf{ps}}$ other than the syntactic relationship that is clear by definition of these logics. An idea for future research might be to extend the Ehrenfeucht-Fraïssé games established for HCTL in Section 5.1 to $\text{HCTL}^*_{\mathsf{ps}}$ and $\text{HCTL}^*_{\mathsf{pp}}$ to have a better framework for reasoning about their expressive power. Another idea which might help in separating both logics might be to find a suitable notion of bisimulation for $\text{HCTL}^*_{\mathsf{ps}}$. We know that the notion of k-bisimulations introduced in Section 4.2 is not enough but have not yet looked at suitable alternatives for $\text{HCTL}^*_{\mathsf{ps}}$.
Also, there are many connections to other recent extensions of CTL^*, the modal μ-calculus and its fragments. We have briefly looked at some loose connections in Section 3.7 but have not yet studied these connections in more detail. Some of those more expressive connections like QCTL^* could potentially also be of use to better understand the relationship between $\text{HCTL}^*_{\mathsf{pp}}$ and $\text{HCTL}^*_{\mathsf{ps}}$.
Another open field of research within hybrid branching-time logics (and hybrid logics in general) is the search for fragments with a decidable satisfiability problem or even one step further, a satisfiability problem with a reasonable computional complexity. Frankly, almost all proposed decidable fragments impose such harsh restrictions on the binder specifically that its presence feels almost negligibile or trivial. Thus, finding more meaningful decidable fragments in which the binder still has some use would be quite interesting.
And finally, we have established some rather loose connections to other extensions of branching-time logics in Section 3.7 and 4.1. One could evaluate those connections and see if they can be used to transfer some of the obtained results like lower or upper bounds or the model theory to other connected fields of research.

Bibliography

[1] M. Adler and N. Immerman. An n! lower bound on formula size. *ACM Trans. Comput. Logic*, 4(3):296–314, July 2003.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[3] B. Aminof, A. Murano, and S. Rubin. CTL* with graded path modalities. *Inf. Comput.*, 262(Part):1–21, 2018.

[4] H. R. Andersen. A polyadic modal μ-calculus. Technical Report ID-TR: 1994-195, Dept. of Computer Science, Technical University of Denmark, Copenhagen, 1994.

[5] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 307–321. Springer Berlin Heidelberg, 1999.

[6] C. Areces, P. Blackburn, and M. Marx. The computational complexity of hybrid temporal logics. *Logic J. of the IGPL*, 8(5):653–679, 2000.

[7] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: Characterization, interpolation and complexity. *The J. of Symbolic Logic*, 66(3):pp. 977–1010, 2001.

[8] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 2001.

[9] S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[10] M. R. F. Benevides and L. M. Schechter. Decidability of a syntactic fragment of the hybrid computation tree logic with the $\downarrow$ operator. 2008.

[11] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for CTL*. In *Proc. 10th Symp. on Logic in Computer Science, LICS'95*, pages 388–397, San Diego, CA, USA, 1995. IEEE.

[12] A. Bianco, F. Mogavero, and A. Murano. Graded computation tree logic. *ACM Trans. Comput. Logic*, 13(3), 2012.

[13] P. Blackburn. Representation, reasoning, and relational structures: A hybrid logic manifesto. *Logic J. of the IGPL*, 8(3):339–365, 2000.

[14] P. Blackburn and M. Tzakova. Hybrid languages and temporal logic. *Logic J. IGPL*, 1999.

[15] L. Bozzelli and R. Lanotte. Complexity and succinctness issues for linear-time hybrid logics. *Theor. Comput. Sci*, 411(2):454–469, 2010.

[16] J. Bradfield and I. Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer International Publishing, Cham, 2018.

[17] F. Bruse, O. Friedmann, and M. Lange. On guarded transformation in the modal μ-calculus. *Logic Journal of the IGPL*, 23(2):194–216, 2015.

[18] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, CA, USA, 1962. Stanford University Press.

[19] R. A. Bull. An approach to tense logic. *Theoria*, 36(3):282–300, 1970.

[20] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263, 2017.

[21] A. K. Chandra, D. C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[22] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *Proc. 2nd Conf. on Computer Aided Verification, CAV'90*, volume 531 of *LNCS*, pages 233–242. Springer, 1991.

[23] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *TCS*, 126(1):77–96, 1994.

[24] S. Demri, V. Goranko, and M. Lange. *Temporal Logics in Computer Science*, volume I – Finite State Systems of *Cambridge Tracts in Theor. Comp. Sc.* Cambridge Univ. Press, 2016.

[25] H. D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Math. Logic. Springer, Berlin, 1995.

[26] H. D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Undergraduate Texts in Mathematics. Springer New York, 1996.

[27] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

[28] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[29] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. of Comp. and Sys. Sc.*, 30:1–24, 1985.

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

[31] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 368–377, 1991.

[32] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th Conf. on Computer Aided Verification, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[33] E. A. Emerson and C. L. Lei. Modalities for model checking: branching time logic strikes back. *Science of Computer Programming*, 8(3):275 – 306, 1987.

[34] J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, SPIN 2017, pages 112–121. ACM, 2017.

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

[36] G. Fontaine, F. Mogavero, A. Murano, G. Perelli, and L. Sorrentino. Cycle detection in computation tree logic. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016*, volume 226 of *EPTCS*, pages 164–177, 2016.

[37] M. Franceschet and M. de Rijke. Model checking hybrid logics (with an application to semistructured data). *Journal of Applied Logic*, 4(3):279 – 304, 2006. Methods for Modalities 3 (M4M-3)Methods for Modalities Workshop 3 (M4M-3).

[38] M. Franceschet, M. de Rijke, and B.-H. Schlingloff. Hybrid logics on linear structures: Expressivity and complexity. In *Proc. 10th Int. Symp. on Temporal Representation and Reasoning, TIME'03, and 4th Int. Conf. on Temporal Logic, ICTL'03*, pages 166–173. IEEE, 2003.

[39] O. Friedmann and M. Lange. Solving parity games in practice. In *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Proceedings*, pages 182–196, 2009.

[40] G. Gargov and V. Goranko. Modal logic with names. *Journal of Philosophical Logic*, 22(6):607–636, 1993.

[41] V. Goranko. Temporal logic with reference pointers. In Dov M. Gabbay and Hans Jürgen Ohlbach, editors, *Temporal Logic*, pages 133–148, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[42] V. Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.

[43] V. Goranko. Temporal logics with reference pointers and computation tree logics. *Journal of Applied Non-Classical Logics*, 10(3-4):221–242, 2000.

[44] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1998.

[45] J. Y. Halpern and Y. Shoham. A propositional model logic of time intervals. In *Proceedings of the Symposium on Logic in Computer Science (LICS '86)*, pages 279–292, 1986.

[46] D. Harel. Recurring dominoes: making the highly undecidable highly understandable. In Marek Karpinski, editor, *Foundations of computation theory: proceedings of the 1983 Inernational FCT-Conference*, vol-

ume 158 of *Lecture notes in computer science*, Borgholm, Sweden, 1983. Springer-Verlag.

[47] M. Hofmann and M. Lange. *Automatentheorie und Logik*. eXamen.press. Springer, 2011.

[48] M. Jurdziński. Deciding the winner in parity games is in $UP \cap$co-UP. *IPL: Information Processing Letters*, 68(3):119–124, 1998.

[49] M. Jurdzinski. Small progress measures for solving parity games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 290–301, 2000.

[50] M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *Proceedings of the Thirty second Annual IEEE Symposium on Logic in Computer Science (LICS 2017)*, pages 1–9. IEEE Computer Society Press, June 2017.

[51] M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[52] A. Kara, V. Weber, M. Lange, and T. Schwentick. On the hybrid extension of CTL and CTL+. In Rastislav Královič and Damian Niwiński, editors, *Mathematical Foundations of Computer Science 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 427–438. Springer Berlin Heidelberg, 2009.

[53] D. Kernberger and M. Lange. Model checking for the full hybrid computation tree logic. In *Proc. 23rd Int. Symp. on Temporal Representation and Reasoning, TIME'16*, pages 31–40. IEEE Computer Society, 2016.

[54] D. Kernberger and M. Lange. The fully hybrid mu-calculus. In *Proc. 24th Int. Symp. on Temporal Representation and Reasoning, TIME'17*, volume 90 of *LIPIcs*, pages 17:1–17:16. Dagstuhl-Leibniz-Zentrum, 2017.

[55] D. Kernberger and M. Lange. Model checking for hybrid branching-time logics. *Journal of Logical and Algebraic Methods in Programming*, 2018.

[56] D. Kernberger and M. Lange. On the expressive power of hybrid branching-time logics. In *25th International Symposium on Temporal Representation and Reasoning, TIME 2018*, pages 16:1–16:18, 2018.

[57] D. Kernberger and M. Lange. On the expressive power of hybrid branching-time logics. *Theoretical Computer Science*, 2019. Submitted for publication.

[58] S. C. Kleene. *Introduction to metamathematics.* 1952.

[59] O. Kupferman and M. Y. Vardi. Memoryful branching-time logic. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 265–274, 2006.

[60] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2), 2000.

[61] M. Lange. A purely model-theoretic proof of the exponential succinctness gap between CTL+ and CTL. *Information Processing Letters*, 108(5):308 – 312, 2008.

[62] M. Lange. Model checking for hybrid logic. *Journal of Logic, Language and Information*, 18(4):465–491, 2009.

[63] M. Lange. The arity hierarchy in the polyadic μ-calculus. In *Proceedings Tenth International Workshop on Fixed Points in Computer Science, FICS 2015*, pages 105–116, 2015.

[64] M. Lange and E. Lozes. Model checking the higher-dimensional modal μ-calculus. In *Proc. 8th Workshop on Fixpoints in Computer Science, FICS'12*, volume 77 of *Electr. Proc. in Theor. Comp. Sc.*, pages 39–46, 2012.

[65] M. Lange and C. Stirling. Model checking games for branching time logics. *Journal of Logic and Computation*, 12(4):623–639, 2002.

[66] F. Laroussinie and N. Markey. Quantified CTL: Expressiveness and complexity. *Logical Methods in Computer Science*, 10, 2014.

[67] F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking CTL$^+$ and FCTL is hard. In *Proc. 4th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS'01*, volume 2030 of *LNCS*, pages 318–331. Springer, 2001.

[68] K. Lehtinen. A modal µ perspective on solving parity games in quasi-polynomial time. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 639–648. ACM, 2018.

[69] M. Marx. Narcissists, stepmothers and spies. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the 2002 International Workshop on Description Logics (DL2002)*, volume 53. CEUR-WS.org, 2002.

[70] A. Meier, M. Mundhenk, T. Schneider, M. Thomas, V. Weber, and F. Weiss. The complexity of satisfiability for fragments of hybrid logic – part I. *J. Applied Logic*, 8(4):409–421, 2010.

[71] S. Miyano and T. Hayashi. Alternating finite automata on -words. *Theoretical Computer Science*, 32(3):321 – 330, 1984.

[72] F. Moller and A. Rabinovich. Counting on CTL*: on the expressive power of monadic path logic. *Information and Computation*, 184(1):147 – 159, 2003.

[73] D. D. Monica, V. Goranko, A. Montanari, and G. Sciavicco. Interval temporal logics: a journey. *Bulletin of the EATCS*, 105:73–99, 2011.

[74] M. Mundhenk, T. Schneider, T. Schwentick, and V. Weber. Complexity of hybrid logics over transitive frames. *J. Applied Logic*, 8(4):422–440, 2010.

[75] M. Otto. Bisimulation-invariant PTIME and higher-dimensional μ-calculus. *Theor. Comput. Sci.*, 224(1–2):237–265, 1999.

[76] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

[77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[78] A. N. Prior. Modality and quantification in S5. *The Journal of Symbolic Logic*, 21(1):60–62, 1956.

[79] K. Reinhardt. The complexity of translating logic to finite automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata Logics, and Infinite Games: A Guide to Current Research*, pages 231–238. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[80] U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *Proc. 1st Int. Joint Conf. on Automated Reasoning, IJCAR'01*, volume 2083 of *LNCS*, pages 76–91. Springer, 2001.

[81] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. of Comp. and Sys. Sc.*, 4:177–192, 1970.

[82] S. Schewe. Solving parity games in big steps. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, Proceedings*, pages 449–460, 2007.

[83] T. Schwentick and V. Weber. Bounded-variable fragments of hybrid logics. In *Proc. 24th Annual Symp. on Theoretical Aspects of Computer Science, STACS'07*, volume 4393 of *LNCS*, pages 561–572. Springer, 2007.

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

[85] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[86] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1 – 22, 1976.

[87] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Mathematics*, 5(2):285–309, June 1955.

[88] B. ten Cate and M. Franceschet. Guarded fragments with constants. *Journal of Logic, Language and Information*, 14(3):281–288, 2005.

[89] B. ten Cate and M. Franceschet. On the complexity of hybrid logics with binders. In Luke Ong, editor, *Computer Science Logic*, volume 3634 of *Lecture Notes in Computer Science*, pages 339–354. Springer Berlin Heidelberg, 2005.

[90] T. van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In D. Beyer and M. Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 291–308. Springer International Publishing, 2018.

[91] P. van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker, Inc., 1997.

[92] M. Y. Vardi and L. J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 240–251. ACM, 1985.

[93] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. 1st Symp. on Logic in Computer Science, LICS'86*, pages 332–344. IEEE, Washington, DC, 1986.

[94] J. Vöge and M. Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *Computer Aided Verification, 12th International Conference, CAV , Proceedings*, pages 202–215, 2000.

[95] P. A. Walega. On expressiveness of Halpern-Shoham logic and its horn fragments. In *24th International Symposium on Temporal Representation and Reasoning, TIME 2017*, 2017.

[96] V. Weber. Hybrid branching-time logics. *CoRR*, abs/0708.1723, 2007.

[97] V. Weber. Branching-time logics repeatedly referring to states. *Journal of Logic, Language and Information*, 18(4):593–624, 2009.

[98] T. Wilke. CTL+ is exponentially more succinct than CTL. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 110–121, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.