Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor's Thesis

# A Proof System for HyperLTL

*Author:*
Norine Coenen

*Advisor:*
Prof. Bernd Finkbeiner, Ph.D.

*Reviewers:*
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr.-Ing. Holger Hermanns

Submitted: 12th February 2016

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

## Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

## Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

## Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 12th February 2016

# Abstract

HyperLTL is a temporal logic defined as an extension of the well known Linear-time temporal logic (LTL).

In LTL it is possible to express properties of single execution traces of a system. This is the reason why LTL is commonly used as the specification language for verification and synthesis tools. Synthesis describes the problem of finding a system that guarantees to satisfy the specification provided as an LTL formula whereas in verification it is checked whether every possible execution trace of a *given* system fulfills the specification. Both techniques aim to prove systems correct which is highly desirable especially in security relevant scenarios.

Although LTL formulas do implicitly reference all paths of a system it is not possible to specify relations between multiple paths explicitly. However, there are some interesting properties like noninterference and observational determinism that are only expressible if several paths can be referenced independently. Properties requiring this additional power are also called hyperproperties. Many interesting information flow security policies belong for instance to the set of hyperproperties.

HyperLTL is obtained from LTL by adding explicit path quantifiers. This already allows the expression of many interesting hyperproperties.

In this thesis we will develop a deductive proof system for the $\forall^*$, the $\exists^*$ and the $\forall^*\exists^*$ fragments of HyperLTL based on the proof systems of CTL$^*$ and ATL. This will make it possible to prove HyperLTL properties of general infinite-state systems, which are beyond the scope of current model checking techniques. Moreover, we will prove that the proof system is sound and for finite-state systems we will achieve relative completeness.

# Acknowledgements

First of all I would like to thank my advisor Prof. Finkbeiner for the opportunity to write my bachelor's thesis at his chair and for the great support during that time. The regular meetings and discussions were really helpful and the support beyond the supervision of my thesis guided me towards a scientific career.

Moreover, I would like to thank Prof. Dr. Hermanns for reviewing this thesis and for helping me to find my way at Saarland University.

I would also like to thank the whole Reactive Systems Group at Saarland University for fully integrating me into the group. Especially I want to thank Hazem for sharing his office and tea with me.

Then I would like to thank my family and friends for all their support. For the scope of this thesis a special thanks goes to Chris for the great discussions and the constant motivation.

I am thankful for the opportunity to make the experience of writing my bachelor's thesis here in Saarbrücken and I know that I learned a lot during that time. Thanks to everyone who contributed to this.

# Contents

# Chapter 1

# Introduction

In computer science studying logics is a very old and central field. As computers today become more and more important in our daily lives, computer science evolves very fast and new fields emerge. For example, in human computer interaction ubiquitous computing is a big topic. This term was coined by Mark Weiser who predicted in the 1990s that in the 21st century everybody will have several computers that will be perfectly integrated into our everyday lives and thus be nearly invisible to humans [28].

With computers being ubiquitous, used effortlessly and partly unconsciously on a daily basis, questions of security and privacy naturally occur. A lack of security might have severe effects, especially for computers used in security-relevant settings like cars or airplanes. In order to prevent computer-caused accidents, proving the correctness of the software used is a desirable competence.

In a first step, it is necessary to formalize what the system is supposed to do. This is called the *specification* the system should fulfill and it is traditionally expressed as a logical formula.

Additionally the system itself has to be modeled. Mostly we consider *reactive systems*, i.e. systems that continually interact with their environment. These systems rarely terminate but are supposed to run infinitely long. A typical example of a reactive system is a vending machine which runs forever and is required to properly react to the orders of the people in its environment. Usually transition systems are used as a model for this kind of system. Such a transition system represents the different states, the reactive system can be in and the transitions between these states. The states of the transition system are labeled. When running, the behavior of the reactive system defines an infinite sequence of states (called *path*) through the transition system. This path defines an infinite sequence of labels (called *trace*) when considering the labels of every state in the path.

A transition system can be verified against a specification by using model checking, a verification technique introduced by E. M. Clarke and E. A. Emerson in 1981 [8]. This technique checks if every path through the system model satisfies the specification. If this is the case the system guarantees to show the desired behavior.

Another even older approach to the design of correct systems was introduced by Church in 1963 and is called synthesis [7]. Here a system model is generated automatically by the synthesis tool according to the input specification. The resulting system is guaranteed to be correct in the sense that it fulfills the specification. If no such system exists the synthesis tool does not output any system model.

Both approaches have in common that the specification describing the desired system behavior has to be formulated mathematically precise. Since 1977 temporal logics provide a suitable formalism to do so. In that year Amir Pnueli introduced Linear-time Temporal Logic (LTL) [19]. Today LTL and its related logics Computational Tree Logic (CTL) [8] and CTL* [12] are widely used as specification languages for verification and synthesis tools. They already suffice to express a wide range of desired system properties and thus allow the formal verification using these specifications as inputs.

However, especially in the area of privacy and security requirements there are many properties that cannot be expressed using these logics. Observational determinism is one of these properties [9]. It requires a system containing high and low security components to behave as a deterministic function from the low security inputs to the low security outputs from every low security user's point of view. This ensures that no information contained in the high security components is leaked to a low security user. Whenever sensitive information should be kept secret in a system, although many users have access to it, this requirement is crucial. Consider for example a financial institution. None of the customers wants their account balance to be leaked to any unauthorized person.

Observational determinism cannot be expressed in LTL because it requires to establish a relation between two different system paths at a time. Neither LTL nor CTL* are expressive enough to specify relations between multiple explicitly referenced paths. The formulas of both logics can only reason about one path at a time. Hence, verification of secrecy constraints is limited when using these logics as specification languages. In order to enable the verification of properties describing relations between multiple paths we need to use another specification language that allows the formalization of these properties.

By extending LTL and CTL* with explicit path quantifiers we gain the temporal logics HyperLTL and HyperCTL* respectively [9]. In formulas of these logics it is possible to directly address the different quantified paths and thus to express desired relations between them. The following HyperLTL formula expresses observational determinism [14]:

$$\forall \pi . \forall \pi' . \Box (\bigwedge_{a \in L} a_\pi \leftrightarrow a_{\pi'}) \rightarrow \Box (\bigwedge_{a \in O} a_\pi \leftrightarrow a_{\pi'})$$

where $L \subseteq AP$ contains all low security input variables and $O \subseteq AP$ is defined to be the set of all low security output variables. The formula requires that every pair of traces in a system with the same values for the low security input variables also has to have the same low security outputs along the whole traces. This means that the system behaves like a deterministic function in the view of a low security user.

Desired system behavior like observational determinism needs the explicit quantification over paths to be formally expressed in order to define relations between these different paths. Every specification that compares and reasons about more than one path defines a so called hyperproperty [14]. Trace properties in contrast only describe the observable characteristics of a single path and are therefore less powerful. However, every trace property can also be expressed as a hyperproperty by simply using only one quantified path. Using a logic for hyperproperties like HyperLTL or HyperCTL* as the specification language for verification and synthesis tools would thus enable the verification of a wider range of desired system performances including many privacy and security requirements.

A deductive proof system for a logic allows to derive new theorems from the axioms of the logic. It offers the possibility to find sound deductive proofs for new theorems by applying the inference rules of this proof system. Having a proof system for a logic is therefore very beneficial and allows to examine the logic more closely. In this thesis we will develop such a deductive proof system for HyperLTL. The logic and the proof system presented are related to other temporal logics and their proof systems. Both will be reviewed briefly in the following section.

**Related Work**

Finkbeiner and Rabe classified the logics for hyperproperties into the field of temporal logics in [14]. They examine the logics LTL, CTL*, HyperLTL and HyperCTL* regarding their ability to express hyperproperties and linear- or branching-time properties.

LTL operates, as its name already suggests, on the linear-time side. It was introduced in 1977 by Amir Pnueli [19] and can express linear-time trace properties that specify the behavior of a single trace. In 1986 the logic CTL* that subsumes LTL was defined by Emerson and Halpern [12]. This temporal logic can express every branching-time trace property. CTL is another sublogic of CTL* introduced by Clarke et al. in 1982 [8]. Like CTL*, it expresses branching-time properties but it is not expressive enough to define every trace property.

Nevertheless, none of the above logics is expressive enough to formalize any hyperproperty. This holds even for CTL* which subsumes both LTL and CTL. Although CTL* offers special path quantifiers, its formulas can only reason about one implicitly referenced path at a time. Therefore, it is impossible to specify any explicit relations between multiple paths because these different paths cannot be addressed.

In order to be able to express hyperproperties, it is essential to be able to reference different paths in the formula. This is achieved by Clarkson, Finkbeiner et al. [9] by introducing explicit path quantifiers that bind path variables. Hence, HyperLTL and HyperCTL* are obtained as extensions of LTL and CTL* respectively. Both logics can express hyperproperties because explicitly quantified paths can be referenced within the formula. The desired relation between these quantified paths can then be formalized using the path variables. According to their origin, HyperLTL and HyperCTL* integrate similarly to LTL and CTL* into the linear- vs. branching-time range. HyperLTL can formalize linear-time hyperproperties and HyperCTL* is able to express all hyperproperties, the linear-time as well as the branching-time ones.

Alternating-time Temporal Logic (ATL) was introduced by Alur et al. in 2002 [3]. Its system model is a game structure with different players. Like CTL*, ATL* offers quantification over all computation paths through the model. However it uses the selective quantifier $\langle\langle A' \rangle\rangle$ where $A'$ is a subset of the set of all players $A$. The formula $\langle\langle A' \rangle\rangle\varphi$ states that the players in $A'$ have a set of strategies to ensure the satisfaction of $\varphi$ regardless of the behavior of the players in $A \setminus A'$. By using $\langle\langle \emptyset \rangle\rangle$ and $\langle\langle A \rangle\rangle$ the universal and existential quantification can be expressed respectively by using the selective quantifier. This shows that ATL* is a generalization of CTL*.

The playful aspect of the ATL* semantic can also be found in HyperLTL. Consider a formula of the following form:

$$\forall \pi_1 . \exists \pi_2 . \varphi$$

This formula requires that path $\pi_2$ can always be chosen adaptively to the choice of $\pi_1$ such that $\varphi$ is fulfilled. The actual selection of $\pi_2$ can be seen as applying a strategy that chooses a path $\pi_2$ when given a choice for $\pi_1$. If the formula $\forall\pi_1.\exists\pi_2.\varphi$ is satisfied then the strategy can always give a correct path choice for $\pi_2$.

Because of these semantical similarities we can benefit from a proof system for ATL* while constructing the proof system for HyperLTL. In 2006 Slanina presented the ATL* proof system on which the HyperLTL proof system will be based [23]. This proof system uses automata-theoretic results to reduce the proofs of arbitrary ATL* properties to proofs in the underlying assertion language [23].

For LTL and CTL* proof systems have been developed earlier. In 2002 a relatively complete deductive proof system for CTL* was presented by Kesten and Pnueli [20]. A sound and relatively complete proof system for LTL was proposed about 10 years earlier in 1991 by Manna and Pnueli [17]. It was the first that completely reduced temporal reasoning to assertional reasoning [17].

**Contribution**
To contribute to the field of formal verification we will develop a sound proof system for the $\forall^*\exists^*$ fragment of the temporal logic HyperLTL within this thesis. For finite state systems and safety formulas, the proof system will achieve completeness relative to the underlying assertion language.

HyperLTL can formalize more than trace properties because the explicit trace quantification allows to express all linear-time hyperproperties. Using HyperLTL as the specification language of verification and synthesis tools thus allows to formally prove every specification expressible in HyperLTL.

There are many interesting privacy and security requirements that can be formalized as a hyperproperty using the logic HyperLTL. For proving that these properties hold on a given system the presented proof system can be used. For example, it is possible to formally verify that a given system does not leak any sensitive data. This is a highly desirable requirement every application in any security-relevant setting should fulfill. Being able to formally prove that a system meets this specification will influence how much users trust this application. Hence, the presented proof system is very beneficial when it comes to model checking security requirements and to verify information-flow security policies like the absence of data leakage.

The proof system presented in this thesis provides valuable insights into the formal verification of hyperproperties. These insights will help finding new verification methods and algorithms for proving hyperproperties in general and information security requirements in particular.

**Overview**

The remaining part of this thesis is structured as follows. In chapter 2 we will give an introduction to the basic concepts used throughout this work. We will present different computational models and temporal logics before introducing Büchi automata and some standard properties of proof systems. In chapter 3 we will develop the proof system for HyperLTL and show that it is sound but not complete. In order to achieve relative completeness of the proof system we introduce an additional proof rule in chapter 4 and show that with this rule the proof system is complete for finite-state systems and safety properties. Chapter 5 will give a summary of this thesis present an outlook on future work.

# Chapter 2

# Preliminaries

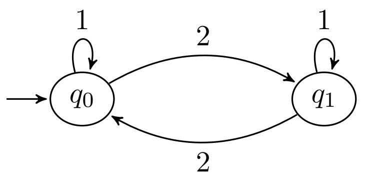
In the following we will introduce the basic concepts needed to understand this thesis. We will start by presenting the computational models on which we can evaluate logical formulas. Afterwards, we formally introduce the temporal logics we will work with using the corresponding computational models. Then a brief introduction to Büchi automata will be given before we will discuss properties of proof systems in general.

## 2.1 The Computational Model

**Kripke Structures with Transition Identifiers**
Kripke structures are commonly used to model closed systems [3]. Closed systems are systems whose behavior only depends on the system but not on for example its environment. The formulas of LTL and HyperLTL are defined over the traces of a Kripke structure.

A *Kripke structure with transition identifiers* $K = (Q, q_0, d, \delta, AP, l)$ consists of the following components:

- $Q$ is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $d : Q \to \mathbb{N}^*$ gives the number of outgoing transitions for every state $q \in Q$. The transitions starting in state $q$ are identified by the numbers $1, \ldots, d(q)$.
  Every state in $Q$ has to have at least one outgoing transition, i.e. $\forall q \in Q.\ d(q) \geq 1$ to ensure that every path can be extended to infinite length.



with $l(q_0) = \{a\}$ and $l(q_1) = \{b\}$.

Figure 2.1: Kripke structure $K$

- $\delta : Q \times \mathbb{N}^* \to Q$ is the transition function yielding a unique successor state when given a state and the identifier of a particular transition. $\delta(q, i)$ is defined for all outgoing transitions $i$ of every state $q \in Q$: $\forall q \in Q.\ \forall\ 1 \leq i \leq d(q).\ \exists s \in Q.\ \delta(q, i) = s$ where $s$ denotes the successor state of $q$ when taking the transition identified by $i$.
- $AP$ is the set of atomic propositions.
- $l : Q \to 2^{AP}$ is the labeling function assigning a set of atomic propositions to every state in $Q$.

The transition identifiers are introduced to the Kripke structures in order to allow a straightforward transition from Kripke structures to concurrent game structures which will be defined subsequently. In the following we will only consider Kripke structure with transition identifiers and simply refer to them as *Kripke structures*.

**Example 2.1 (Kripke Structure)**
Kripke structures are usually presented graphically as in Figure 2.1 instead of using the tuple notation directly.

The states are labeled with their names from the set $Q$ and the labeling function is written underneath the Kripke structure. The initial state is marked with an incoming edge without any predecessor. The transitions are represented via edges connecting every state with its successor state. For every state $q$ $d(q)$ is exactly the out-degree of state $q$. We identify the outgoing transitions of a state $q$ by the numbers $1, \ldots, d(q)$ in clockwise direction starting at the top as illustrated above. $\square$

When working with Kripke structures we often argue about the paths or traces of these structures. In the following these terms are defined.

A *path* $p = s_0 s_1 s_2 s_3 \ldots$ of a Kripke structure is an infinite sequence of states where the first state has to be the initial state, i.e. $s_0 = q_0$ and $\forall i \in \mathbb{N}.\ \exists\ 1 \leq j \leq d(s_i).\ \delta(s_i, j) = s_{i+1}$. In the example above $p = q_0 q_0 q_1 q_0 (q_1)^\omega$ is a path of this Kripke structure. The set of all paths of a Kripke structure $K$ starting in state $s$ is denoted by $Paths(K, s)$. $Paths^*(K, s)$ also contains all suffixes of these paths. This means that for every path $p$ in $Paths(K, s)$ and every natural number $i$ there is a path $p'$ in the set $Paths^*(K, s)$ which is equal to $p$ with the first $i$ states deleted.

A *trace* $t \in (2^{AP})^\omega$ of a Kripke structure is an infinite sequence of atomic propositions generated by a path, i.e. $\forall i \in \mathbb{N}.\ t_i = l(s_i)$ for a path $p = s_0 s_1 s_2 s_3 \dots$ through the Kripke structure. The trace $t$ of path $p$ for our example is $\{a\}\{a\}\{b\}\{a\}(\{b\})^\omega$. The set $Tr(K, s)$ represents the set of all traces of paths in $Paths(K, s)$ and $Tr^*(K, s)$ denotes all their suffixes.

Now we will introduce some important operations on traces and paths.

Given a trace (or path) $t$ and a natural number $i$, we write $t[i]$ to denote the $i$-th element of $t$. In the special case where $i = 0$, $t[0]$ yields the set of atomic propositions that hold in the first state of the corresponding path.

When we are only interested in a certain part of a trace, we write $t[i, j]$ for natural numbers $i$ and $j$ to access the partial trace $t_i t_{i+1} \dots t_{j-1} t_j$. $t[0, i]$ thus gives the finite prefix of $t$ ending with element $i$ and $t[i, \infty]$ represents the infinite suffix of $t$ starting with element $i$.

**Concurrent Game Structures**

Concurrent game structures are suitable to model compositions of open systems. Open systems are systems whose behavior is dependent on the environment and the system itself [3]. The semantic of ATL* is defined over concurrent game structures.

A *concurrent game structure* $S = (A, Q, AP, l, d, \delta)$ is defined by the following components:

- $A$ is a set of players (or agents).
- $Q$ is a finite set of game states.
- $AP$ is a set of atomic propositions.
- $l : Q \to 2^{AP}$ is a labeling function assigning a set of atomic propositions to every state.
- $d_{p_l} : Q \to \mathbb{N}^*$ gives for every state $q \in Q$ and every player $p_l \in A$ with $1 \leq l \leq |A|$ the number of possible moves. To ensure the infinite continuation of every path, it is required that in every state every player has at least one possible move. The different moves of $p_l$ in $q$ are identified by the numbers $1, \dots, d_{p_l}(q)$.
  If we consider a state $q$ with exactly one move $j_{p_l}$ per player $p_l$ we obtain a move vector $(j_{p_1}, \dots, j_{p_{|A|}})$, where $\forall 1 \leq l \leq |A|.\ 1 \leq j_{p_l} \leq d_{p_l}(q)$ holds. $D(q) = \{1, \dots, d_{p_1}(q)\} \times \dots \times \{1, \dots, d_{p_{|A|}}(q)\}$ is the set of all move vectors for state $q$ and $D$ is called move function.

- $\delta$ is the transition function that takes a state $q \in Q$ and a move vector from the set $D(q)$ and yields the successor state $s \in Q$ that is reached from $q$ if all players choose their moves according to the given move vector. Formally $\forall q \in Q.\ \forall (j_{p_1}, \dots, j_{p_{|A|}}) \in D(q).\ \exists s \in Q.\ \delta(q, (j_{p_1}, \dots, j_{p_{|A|}})) = s$.

Note that Kripke structures are a special case of concurrent game structures where $A$ is a singleton containing only the system player [3].

A computation $\lambda = q_0\, q_1\, q_2\, \dots$ of a concurrent game structure is, like a path of a Kripke structure, an infinite sequence of states where $\forall i \in \mathbb{N}.\ \exists v \in D(q_i).\ \delta(q_i, v) = q_{i+1}$. Given two natural numbers $i$ and $j$ we use $\lambda[i]$ to access the $i$-th element of the computation $\lambda$ and $\lambda[i, j]$ to denote the partial computation $q_i q_{i+1}\, \dots\, q_{j-1} q_j$. These operations are defined analogously to the trace manipulation operations.

The different players of a concurrent game structure may have different objectives. Depending on the choices of all other players one player may have to decide in different ways in order to achieve her goal. These different decisions are compromised in so called strategies.

A strategy $\sigma_{p_l} : Q^+ \to \mathbb{N}^*$ for a player $p_l \in A$ maps every finite prefix of a computation $q_0 \dots q_n$ to a natural number $i = \sigma_{p_l}(q_0 \dots q_n)$ with $1 \leq i \leq d_{p_l}(q_n)$ and $1 \leq l \leq |A|$. Thus the strategy $\sigma_{p_l}$ determines a move $i$ for player $p_l$ on the current state depending on its history. If the strategy does not need the information given by the history but only works on a single state $q_n$, the strategy $\sigma_{p_l} : Q \to \mathbb{N}^*$ is called memoryless.

Let $A' \subseteq A$ be a set of players. We consider the set $F_{A'} = \{\sigma_{p_l} |\ 0 \leq l \leq |A'|\}$ of strategies containing exactly one strategy per player $p_l \in A'$. Using this notation we can define $out(q, F_{A'})$ as the set of computations starting with the state $q$ where all players in $A'$ choose their moves according to their strategies in $F_{A'}$. The choices of the players in $A \setminus A'$ are arbitrary. The players in $A'$ together with their strategies in $F'_A$ can enforce the computations of the set $out(q, F_{A'})$.

## 2.2 Temporal Logics

In computer science logic plays a central role in a wide range of applications. One of them is formal system verification. As already discussed in the introduction, logical formulas are used to formally specify the desired behavior of a given system. It is especially important that these formulas cover all relevant aspects of this behavior with respect to the real world. If it is for example required for a system to answer a request in a given time span, this demand can only be expressed formally if the logic used to specify the desired properties is able to reason about timing constraints. It is only possible to gain a valid result that can be transfered to a real world application if every aspect of a real world requirement can be correctly formalized as a logical formula.

Propositional logic for instance is not able to express a progress in time which is why it is not usable for the verification of temporal properties. Hence we consider appropriate logics that are also able to express time progress. With these logics we can express requirements demanding for example that a certain event will occur infinitely often.

We will start by formally defining Linear-time temporal logic [19] and Quantified Propositional Time Logic [22]. Then Alternating-time temporal logic will be introduced [3] before we finally consider HyperLTL [9].

### 2.2.1 Linear-time Temporal Logic

Linear-time Temporal Logic (LTL) was first introduced by Amir Pnueli in 1977. With LTL he proposed "a unified approach to program verification" [19]. It enables the formal specification of system properties that have constraints regarding the temporal sequence of actions. We will now formally introduce the syntax and semantics of LTL.

**Syntax**
The core idea of LTL is the introduction of temporal operators. With these it is possible to argue about events in the future and how they are related to each other. Every LTL formula is generated by the following grammar where $a$ is an atomic proposition that is either $true$ or $false$:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\ \mathcal{U}\ \varphi$$

Besides the atomic propositions, LTL also has several operators. There are the Boolean connectives *negation* ($\neg$) and *conjunction* ($\wedge$) as well as the usual derived operators *disjunction* $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, *implication* $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$ and *equivalence* $\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. Additionally LTL has the temporal operators *next* ($\bigcirc$) and *until* ($\mathcal{U}$). Two more temporal operators can be derived as follows: $\Diamond\varphi \equiv true\ \mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Intuitively *eventually* ($\Diamond$) means that a formula will become $true$ at some point of time in the future and *globally* ($\Box$) requires that a formula is satisfied at every state in the future.

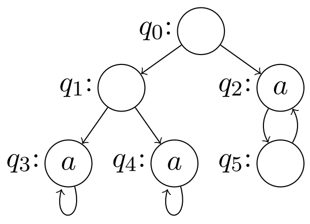


Figure 2.2: Kripke structure $K'$

**Semantics**
We will now define the semantics of LTL formally. A formula $\varphi$ is interpreted over a trace $t$. The relation $\models$ states if a trace satisfies an LTL formula and is defined as follows:

$$\begin{array}{lll} t \models a & \text{iff} & a \in t[0] \\ t \models \neg\varphi & \text{iff} & t \not\models \varphi \\ t \models \varphi_1 \wedge \varphi_2 & \text{iff} & t \models \varphi_1 \text{ and } t \models \varphi_2 \\ t \models \bigcirc\varphi & \text{iff} & t[1,\infty] \models \varphi \\ t \models \varphi_1 \, \mathcal{U} \, \varphi_2 & \text{iff} & \exists\, i \geq 0 : t[i,\infty] \models \varphi_2 \text{ and} \\ & & \forall\, 0 \leq j < i : t[j,\infty] \models \varphi_1 \end{array}$$

An LTL formula $\varphi$ holds on a given Kripke structure $K = (Q, q_0, d, \delta, AP, l)$ if and only if $\forall t \in Tr(K, q_0).\ t \models \varphi$ holds. It is important to note that $\varphi$ is always evaluated on a single trace $t$. LTL formulas can thus only express properties of one trace or in terms of Kripke structures they express properties holding on every trace of this Kripke structure.

**Example 2.2 (LTL)**
Consider the Kripke structure $K'$ from Figure 2.2 where the states are labeled according to the labeling function $l$ and $a$ is written for $\{a\}$. The names of the states can be found on the left of every state.

The formula $\varphi_1 = \Box\Diamond a$ requires a trace to have infinitely many $a$'s. For the Kripke structure $K'$ $\varphi_1$ is satisfied because every trace of $K'$ fulfills the formula, i.e. $\forall t \in Tr(K', q_0).\ t \models \varphi_1$.

In contrast to that the formula $\varphi_2 = \Diamond\Box a$ is not *true* for $K'$. $\varphi_2$ claims that there has to be a point in the future from which on $a$ holds at every state. Now consider the path $q_0(q_2q_5)^\omega$ of $K'$. The corresponding trace $t$ is $\emptyset(\{a\}\emptyset)^\omega$. For this trace $\varphi_2$ is not *true* because elements containing $a$ alternate with elements that do not contain $a$. So $a$ will never hold *globally* and $t \not\models \Diamond\Box a$. $\square$

### 2.2.2 Quantified Propositional Temporal Logic

The Quantified Propositional Time Logic (QPTL) was introduced in 1983 by Sistla in his Doctoral Dissertation [22]. It extends LTL with propositional quantification, i.e. in QPTL it is possible to quantify over atomic propositions.

**Syntax**
The syntax of QPTL is defined by the following grammar where $a$ is an atomic proposition:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \;\mathcal{U}\; \varphi \mid \exists a.\varphi$$

QPTL formulas can contain the same operators as LTL formulas and in addition the propositional quantifier $\exists a$ is introduced. This quantifier requires that there is an additional atomic proposition that can be chosen in a way such that $\varphi$ is fulfilled.

**Semantics**
The semantic of QPTL is the same as the one of LTL for all operators the both logics share. Additionally the semantics for the propositional quantification is defined for QPTL:

$$t \models \exists a.\varphi \;\text{ iff }\; \exists\, t' \in (2^{AP})^{\omega}.\; (t =_{AP\setminus\{a\}} t') \wedge (t' \models \varphi)$$

where $t =_{AP\setminus\{a\}} t'$ states that $t$ and $t'$ are equal at every point in time but without regarding the atomic proposition $a$. This intuitively means that whenever a new proposition is introduced by a propositional quantifier, the trace considered is extended by a new atomic proposition $a$ and the truth values of $a$ can be chosen freely.

With this modification it is possible to formally express more languages than in LTL. Intuitively this is the case because the quantified propositions can be used as markers such that a hidden structure can be described to define the requirements of a complex language. We see this in the following example:

**Example 2.3 (QPTL)**
Consider the QPTL formula $\exists a.\Box(a \leftrightarrow \bigcirc\neg a) \wedge (a \leftrightarrow b)$. This formula describes the traces that alternate between elements in which $b$ holds and elements in which $b$ does not hold. This is done by marking every second state with the hidden atomic proposition $a$. The atomic proposition $a$ is said to be hidden because it does not occur in the traces described by the formula. $\square$

### 2.2.3 Alternating-time Temporal Logic

We can express linear-time trace properties with LTL. Unfortunately linear properties are not always expressive enough. For example we cannot require the existence of a trace in a Kripke structure satisfying some property with any LTL formula because in LTL we have implicit universal quantification and can only reason about all traces of the Kripke structure as a whole. Linear-time logics are moreover unable to distinguish between Kripke structures that have the same set of traces but different branching structures. This is because their formulas are interpreted over traces which have no information about the actual path and the branching behavior of the system. To express properties with existential quantification or reason about the branching structure of a system we have to argue about so called branching-time properties. These properties allow to universally or existentially quantify over all paths of the model rather than just the traces.

The logic CTL* is a temporal logic that is able to express branching-time properties. It was presented by Emerson and Halpern in 1986 [12] and introduces the two path quantifiers $A$ and $E$ for the universal and existential quantification, respectively. Its formulas are either state or path formulas evaluated on a state or a trace of the system. Because CTL* subsumes LTL it is also possible to express the linear-time properties in CTL* by preceding the LTL formula with a universal quantifier. The LTL formula $\Diamond a$ is expressed in CTL* by the formula $A \Diamond a$. Both formulas require that on every trace of the Kripke structure there will eventually occur the atomic proposition $a$. In contrast to that the CTL* formula $E \Diamond a$ requires only the existence of one such path. This cannot be expressed in LTL because of the existential quantification.

Another sublogic of CTL* is the computation tree logic CTL [8]. It has the syntactic restriction that every temporal operator has to have exactly one quantifier directly preceding it. Although LTL and CTL are both subsets of CTL* the two logics are not directly comparable. There are properties that are expressible in LTL but not in CTL and vice versa. The CTL* formula $A \Box \Diamond a$ for example requires that every trace of a system contains infinitely many $a$'s. This can be translated to LTL yielding the formula $\Box \Diamond a$ but it is not expressible in CTL because every temporal operator needs to be quantified on its own. On the other hand the formula $E \Diamond a$ cannot be expressed in LTL as mentioned above but it is contained in CTL.

On systems with more than one party, for example concurrent game structures with multiple players, we still can use universal and existential quantification. Those quantifiers range over the choices of all players taken at the same time. But we might also like to use a more specific kind of quantification that only ranges over a subset of all players and their respective choices. In that way it is possible to demand that a subset of players has a strategy to achieve a certain goal regardless of what every other player does.

This kind of quantification is called selective and is used in the alternating-time temporal logics ATL and ATL* introduced by Alur et al. in 1997 [3]. Both logics work on concurrent game structures and the selective quantification is realized by the new quantifier $\langle\!\langle A' \rangle\!\rangle$ where $A'$ is a subset of the set of all players $A$ and selection is done over the set of players. ATL and ATL* are generalizations of CTL and CTL* from the branching-time to the alternating-time setting. In ATL* arbitrary nesting of quantifiers and temporal operators is allowed while in ATL every temporal operator has to be directly quantified. We will formally introduce the syntax and semantics of ATL* (and thus implicitly for ATL).

**Syntax**
ATL* formulas can either be state or path formulas. In the following definition let $a$ be an atomic proposition and $A'$ be an arbitrary set of players. Then state formulas are of the following form:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\!\langle A' \rangle\!\rangle\ \psi$$

where $\psi$ is a path formula defined by the following grammar:

$$\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\ \psi \mid \psi\ \mathcal{U}\ \psi$$

where $\varphi$ is a state formula.

For both, state and path formulas we can derive the Boolean operators *conjunction* ($\wedge$), *implication* ($\rightarrow$) and *equivalence* ($\leftrightarrow$) with the use of the *negation* ($\neg$) and *disjunction* ($\vee$) in the same way as for LTL. Analogously to LTL the temporal operators *globally* ($\Box$) and *eventually* ($\Diamond$) can be defined for path formulas using *next* ($\bigcirc$) and *until* ($\mathcal{U}$).

The operator $\langle\!\langle A' \rangle\!\rangle$ denotes the selective path quantifier. It requires that the players in $A'$ together have a set of strategies, one for each player in $A'$, to ensure that the quantified formula is satisfied. Note that the players in $A'$ have to choose their moves before the players in $A \setminus A'$ such that the current choice of the unquantified players is unknown to the players in $A'$ in every step.

We define the dual selective path quantifier $[\![A']\!]\varphi := \neg\langle\!\langle A' \rangle\!\rangle\neg\varphi$ intuitively meaning that there is no set of strategies for the players in $A'$ that guarantees that the formula $\varphi$ will not be fulfilled. The usual existential and universal quantification can be expressed using these path quantifiers with $A$ being the set of all players: $\exists = \langle\!\langle A \rangle\!\rangle$ and $\forall = [\![A]\!]$.

We will often write $\langle\!\langle a_1, \ldots, a_n \rangle\!\rangle$ and $[\![ a_1, \ldots, a_n ]\!]$ instead of $\langle\!\langle \{a_1, \ldots, a_n\} \rangle\!\rangle$ and $[\![ \{a_1, \ldots, a_n\} ]\!]$. An ATL* formula is in *negation normal form* if all negations ($\neg$) in that formula occur directly in front of an atomic proposition.

As mentioned above, ATL is the fragment of ATL* where no arbitrary nesting of quantifiers is allowed, but every temporal operator has to be preceded by exactly one quantifier.

**Semantics**
The semantics of ATL* is defined over concurrent game structures. ATL* state formulas are evaluated in a single state of the concurrent game structure whereas path formulas reason about a path of this system. Let $S = (A, Q, AP, l, d, \delta)$ be a concurrent game structure with $q \in Q$.

$$
\begin{array}{lll}
S, q \models a & \text{iff} & a \in l(q) \\
S, q \models \neg\varphi & \text{iff} & S, q \not\models \varphi \\
S, q \models \varphi_1 \vee \varphi_2 & \text{iff} & S, q \models \varphi_1 \text{ or } S, q \models \varphi_2 \\
S, q \models \langle\!\langle A \rangle\!\rangle \psi & \text{iff} & \exists F_A . \forall \lambda \in \mathit{out}(q, F_A) . \lambda \models \psi \\
S, \lambda \models \varphi & \text{iff} & S, \lambda[0] \models \varphi \text{ for a state formula } \varphi \\
S, \lambda \models \neg\psi & \text{iff} & S, \lambda \not\models \psi \\
S, \lambda \models \psi_1 \vee \psi_2 & \text{iff} & S, \lambda \models \psi_1 \text{ or } S, \lambda \models \psi_2 \\
S, \lambda \models \bigcirc \psi & \text{iff} & S, \lambda[1, \infty] \models \psi \\
S, \lambda \models \psi_1 \;\mathcal{U}\; \psi_2 & \text{iff} & \exists i \geq 0.\ S, \lambda[i, \infty] \models \psi_2 \wedge \forall 0 \leq j < i.\ S, \lambda[j, \infty] \models \psi_1
\end{array}
$$

A set of strategies $F_A$ is called *winning* from state $q$ for the players in $A$ if every computation in $\mathit{out}(q, F_A)$ models the formula $\psi$.

ATL* can describe properties of systems in which several players compete and try to achieve opposing goals. Moreover, it is possible to express properties of the set of computations that a single player or a subset of players can enforce. This allows to quantify much more gradually than simple universal and existential quantification would do and thus enables the expression of more specialized requirements. We will illustrate this in the following example.

**Example 2.4 (ATL* [4])**
Consider the concurrent game structure $S$ from Figure 2.3. The two players $\pi$ and $\pi'$ both choose a move in a state and the two decisions together define the actual transition. Let the first number annotated at the edges be the move chosen by player $\pi$ and the second number represents the choice of player $\pi'$. If both players have chosen their moves, the respective transition is taken.

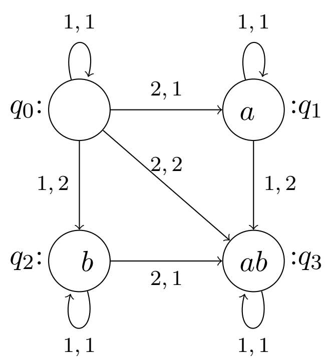


Figure 2.3: The concurrent game structure $S$ with two players $\pi$ and $\pi'$

Player $\pi$ controls the atomic proposition $a$ and $\pi'$ controls $b$ as follows. In case an atomic proposition is $false$ there are two moves available for the respective player. One that changes the value to *true* and the other leaving the value unchanged. If the atomic proposition is already *true* then there is only one move available that does not change the truth value.

The ATL formula $\langle\langle\pi\rangle\rangle\Box\neg a$ requires that player $\pi$ has a strategy to ensure that $a$ will never become *true* whatever move player $\pi'$ chooses. On the concurrent game structure $S$ this formula is *true*, i.e. $S \models \langle\langle\pi\rangle\rangle\Box\neg a$ holds because the value of $a$ is controlled by $\pi$ and it can always choose move number 1 that does not change the value of $a$. This requirement is neither expressible in LTL nor in CTL*.

In LTL we can only reason about all traces of a system without the possibility to only consider a subset of traces. The LTL formula $\Box\neg a$ does not hold on this concurrent game structure if interpreted as a Kripke structure, because there are traces where $a$ is *true* at some position. One of these traces is $\emptyset(\{a\})^\omega$.

In CTL we can at least require the existence of a path on which $a$ never holds. This can be expressed by the formula $E\Box\neg a$ that is indeed *true* because there is for instance the path $q_0^\omega$ whose trace $\emptyset^\omega$ satisfies $\Box\neg a$. In order to satisfy this CTL formula it is sufficient to find one appropriate path. But in CTL* it is not possible to argue about a certain subset of paths and to require some specified behavior for all paths from this subset.

This can be done in ATL* as the above formula shows. We can reason about the sets of computations enforceable by every subset of players allowing us to formalize more precise demands like the one above saying that player $\pi$ alone can ensure that $a$ will never hold.

The formula $[\![\pi]\!] \Diamond b$ is equivalent to $\neg\langle\!\langle\pi\rangle\!\rangle \Box \neg b$ and states that player $\pi$ cannot avoid that eventually the atomic proposition $b$ will be set. This holds on the concurrent game structure $S$ and is due to the fact that player $\pi'$ controls the truth value of $b$ and $\pi$ cannot influence this decisions. $\square$

### 2.2.4 HyperLTL

So far we considered the properties expressible in ATL* or LTL. We saw that ATL* formulas formalize alternating-time properties and LTL formulas can express linear-time trace properties. To see what exactly the term *trace property* means consider a Kripke structure $K$ and an LTL formula $\varphi$. This formula can be evaluated on a single trace of the Kripke structure at a time and for every trace we can decide if this trace satisfies the formula or not. The set of all traces of $K$ that satisfy the given LTL formula $\varphi$ is clearly a subset of all traces $Tr(K, q_0)$ defined by that LTL formula and is called trace property.

**Definition 2.5 (Trace Property, [10])**
For a given Kripke structure $K$ with initial state $q_0$ and the set of all traces in $K$ denoted by $Tr(K, q_0)$, we call every subset of this set of traces a trace property $T_K$ of $K$: $T_K \subseteq Tr(K, q_0)$. $\square$

With these trace properties many interesting requirements can already be formalized. Nevertheless, there are requirements like *observational determinism* mentioned in the introduction that cannot be expressed as a trace property (i.e. as a set of traces) by giving a corresponding LTL formula for example. This is because in this specification it is necessary to examine two traces simultaneously in order to compare their behavior. Trace properties as well as LTL and even CTL* formulas can only argue about a single trace at a time and are thus not expressive enough.

To overcome this limitation and to allow the expression of more general requirements, HyperLTL was recently introduced by Clarkson, Finkbeiner et al. [9]. As the name suggests, it is an extension of LTL. Explicit trace quantifiers are added to LTL in order to be able to describe desired relations between different traces of a system. Before we take a closer look on the expressiveness gained by this extension, we will first formally define HyperLTL by giving its syntax and semantic.

**Syntax**
The syntax of HyperLTL [9] introduces the explicit trace quantifiers $\forall\pi$ and $\exists\pi$. HyperLTL formulas consist of a quantifier prefix which is followed by an LTL formula. Formulas of that form are in *prenex normal form*. In the following grammar defining the syntax, let $a$ be an atomic proposition from

the set $AP$. Every atomic proposition is annotated with a trace variable $\pi \in \mathcal{V}$ indicating on which trace the formula requires $a$ to hold.

$$\begin{aligned}
\psi &::= \exists \pi.\psi \mid \forall \pi.\psi \mid \varphi \\
\varphi &::= a_\pi \quad \mid \neg\varphi \quad \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi
\end{aligned}$$

HyperLTL formulas can use the same operators as LTL formulas. We can thus use the boolean operators *negation* ($\neg$), *disjunction* ($\vee$), *conjunction* ($\wedge$), *implication* ($\rightarrow$) and *equivalence* ($\leftrightarrow$) as well as the temporal operators *next* ($\bigcirc$), *until* ($\mathcal{U}$), *eventually* ($\Diamond$) and *globally* ($\Box$).

Moreover, there are the two trace quantifiers that introduce new trace variables from the set of trace variables $\mathcal{V}$. We only consider closed HyperLTL formulas meaning that every trace variable $\pi$ used as an index of an atomic proposition $a_\pi$ has to be bounded by either the universal trace quantifier ($\forall$) or by the existential trace quantifier ($\exists$). To indicate that the only path variables that are used in a formula $\varphi$ are for example the variables $\pi$ and $\pi'$ from $\mathcal{V}$ we write $\varphi(\pi, \pi')$.

**Semantics**
The semantics of HyperLTL is defined over a Kripke structure $K$ together with a trace assignment $\Pi : \mathcal{V} \mapsto Tr^*(K, q_0)$ that maps a trace variable $\pi$ to a traces $t$ of the Kripke structure $K$. The operation $\Pi[\pi \mapsto t]$ adds the assignment of $\pi$ to $t$ to the trace assignment $\Pi$. If $\pi$ was already assigned to some trace $t'$ in $\Pi$ this assignment is overwritten by the assignment of $\pi$ to $t$. $\Pi(\pi)$ returns the trace that is assigned to the variable $\pi$ in $\Pi$. The known trace operations are applicable to these traces so that for example $\Pi(\pi)[0]$ returns the first element of the trace assigned to $\pi$ and $\Pi(\pi)[i, \infty]$ gives the suffix of that path starting with element $i$. $\Pi[i, \infty]$ modifies the trace assignment $\Pi$ by replacing every trace $t$ with $t[i, \infty]$ such that $\Pi$ returns the corresponding suffix of the trace assigned to $\pi$.

$$\begin{array}{lll}
\Pi \models_K \exists \pi.\psi & \text{iff} & \exists\, t \in Tr(K, q_0) : \Pi[\pi \mapsto t] \models_K \psi \\
\Pi \models_K \forall \pi.\psi & \text{iff} & \forall\, t \in Tr(K, q_0) : \Pi[\pi \mapsto t] \models_K \psi \\
\Pi \models_K a_\pi & \text{iff} & a \in \Pi(\pi)[0] \\
\Pi \models_K \neg\varphi & \text{iff} & \Pi \not\models_K \varphi \\
\Pi \models_K \varphi_1 \vee \varphi_2 & \text{iff} & \Pi \models_K \varphi_1 \text{ or } \Pi \models_K \varphi_2 \\
\Pi \models_K \bigcirc \varphi & \text{iff} & \Pi[1, \infty] \models_K \varphi \\
\Pi \models_K \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \exists\, i \geq 0 : \Pi[i, \infty] \models_K \varphi_2 \text{ and} \\
 & & \forall\, 0 \leq j < i : \Pi[j, \infty] \models_K \varphi_1
\end{array}$$

We will often write $K \models \varphi$ when we mean $\Pi \models_K \varphi$ for a HyperLTL formula $\varphi$ where $\Pi$ is the empty trace assignment $\{\}$.

With HyperLTL we can reference multiple different traces in one formula allowing to define relations between these traces. We are able to formalize the observational determinism example from the introduction for a Kripke structure $K$ with $AP$ as the set of atomic propositions. Let $L \subseteq AP$ be the set of all low security input variables and $O \subseteq AP$ be the set of all low output variables.

$$\forall \pi . \forall \pi' . \square ( \bigwedge_{a \in L} a_\pi \leftrightarrow a_{\pi'} ) \rightarrow \square ( \bigwedge_{a \in O} a_\pi \leftrightarrow a_{\pi'} )$$

The formula requires intuitively that the system $K$ behaves from a low security user's point of view as a deterministic function from the low security inputs to the low security outputs. More precisely it requires every pair of traces that have at every point in time the same low security input values to also have the same low security output values throughout the whole traces expressed by the globally-operator ($\square$).

In HyperLTL it is possible to formally express requirements beyond trace properties. The quantification over different traces allows to reason about the behavior of a set of traces. Thus every HyperLTL formula determines a set of sets of traces that fulfill this formula. Because sets of traces are exactly defined to be trace properties, HyperLTL formulas actually describe a set of trace properties. These sets of trace properties are also called *hyperproperties*.

**Definition 2.6 (Hyperproperty, [10])**
For a given Kripke structure $K$ with initial state $q_0$ and the set of all traces in $K$ denoted by $Tr(K, q_0)$, we call every set of sets of traces from $Tr(K, q_0)$ a hyperproperty $H_K$ of $K$: $H_K \subseteq 2^{Tr(K,q_0)}$. $\square$

In the following example some hyperproperties are examined with respect to a concrete Kripke structure.

**Example 2.7 (HyperLTL)**
Consider again the Kripke structure $K$ from Figure 2.1.

With HyperLTL we can require that for every trace in the system there has to be a corresponding trace that behaves exactly the same but is always one step ahead. The formula $\forall \pi . \exists \pi' . \square(\bigcirc a_\pi \leftrightarrow a_{\pi'})$ expresses exactly this requirement in HyperLTL. The Kripke structure $K$ is no model for that formula because we cannot find a corresponding trace for every trace. To see this consider the trace $t = \{a\}\{b\}\{b\}(\{a\})^\omega$. The HyperLTL formula requires to find a trace $t' = \{b\}\{b\}(\{a\})^\omega$ in $K$ which is not possible because the initial state of $K$ is fixed for every path and in that state $b$ does not hold so no trace of $K$ can start with the atomic proposition $b$ being *true*.

As soon as we ignore the first element of the traces, we can find a corresponding partner for every trace. The trace $t'' = \{a\}\{b\}(\{a\})^\omega$ is then the matching one to the trace $t$. This modified requirement can be expressed by adding a $\bigcirc$ operator to skip the first state resulting in the HyperLTL formula $\forall\pi.\exists\pi'.\bigcirc\square(\bigcirc a_\pi \leftrightarrow a_{\pi'})$. This holds on Kripke structure $K$, i.e. $K \models \forall\pi.\exists\pi'.\bigcirc\square(\bigcirc a_\pi \leftrightarrow a_{\pi'})$.

Instead of ignoring the first state, it is also possible to require the second trace to be exactly the same at every point. The formula $\forall\pi.\exists\pi'.\square(a_\pi \leftrightarrow a_{\pi'})$ expresses this formally and is trivially *true* on every system because as soon as $\pi'$ selects the same trace as $\pi$ does, the equivalence holds immediately and $K \models \forall\pi.\exists\pi'.\square(a_\pi \leftrightarrow a_{\pi'})$. $\square$

Similar to how we extended the linear-time logic LTL to HyperLTL in order to express hyperproperties, we can also gain the temporal logic HyperCTL* from CTL* by adding explicit path quantifiers [14]. HyperCTL* is, as well as CTL*, a branching-time logic which means that it is possible to express hyperproperties that can distinguish systems with different branching structures but the same set of traces. This is due to the fact that the quantifiers in HyperCTL* bind paths instead of traces such that the information about the branching behavior of the system is still accessible. Moreover, the position of the path quantifiers is not limited to the beginning of the formula such that HyperCTL* formulas are not necessarily in prenex normal form. An example of a hyperproperty that is expressible in HyperCTL* but not in HyperLTL can be found in [14]. This shows that HyperCTL* is more expressive that HyperLTL.

## 2.3 Büchi Automata

Automata in general are used to represent the behavior of a system by representing the different system states and the possible transitions between them. Every transition is labeled with certain letters from the alphabet. If these letters match the input, then the corresponding transition may be taken. A sequence of these letters is called a *word* and defines in the deterministic case a unique sequence of states through the system which is called a *run*. If we consider automata for infinite words, then we are left with infinite runs. A run is accepting if it fulfills the requirements specified by the acceptance condition of the automaton. We only consider automata with the *Büchi acceptance condition* which requires a run to visit states from the set of accepting states infinitely often in order to be accepted. If in an

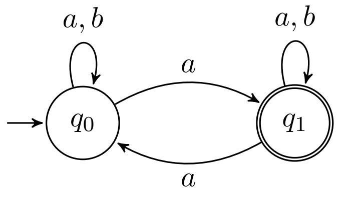


Figure 2.4: Nondeterministic Büchi Automaton $A$

automaton $A$ the run generated by a word $\alpha$ is accepting then we say that this word is in the language of that automaton: $\alpha \in \mathscr{L}(A)$.

In the following we will formally define nondeterministic and alternating Büchi automata. These automata are named after Julius Richard Büchi who first used this concept in 1960 [6] in order to answer Tarski's problem whether the Monadic Second-Order Logic of One Successor (S1S) is decidable. This problem was discussed in [21] by Robinson.

**Nondeterministic Büchi Automata**
A *nondeterministic Büchi automaton* $A = (Q, q_0, \Sigma, \delta, F)$ consists of the following components:

- $Q$ is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $\Sigma$ is a finite alphabet.
- $\delta : Q \times \Sigma \to 2^Q$ is the transition relation, which gives for a state $q \in Q$ and a letter $a \in \Sigma$ a set of possible successor states $Q' \subseteq Q$.
- $F \subseteq Q$ is a set of accepting states.

This automaton is called nondeterministic because there may be more than one possible successor state $q'$ when reading a letter $a$ in state $q$. The actual successor state is then chosen nondeterministically from this set. In contrast to that a *deterministic Büchi automaton* has a *transition function* $\delta' : Q \times \Sigma \to Q$ that outputs a unique successor state on every input from the set $\Sigma$ in a state $q$.

In a nondeterministic automaton $A$ a word $\alpha$ is in the language $\mathscr{L}(A)$ of the automaton if there exists an accepting run of $A$ on $\alpha$, i.e. the nondeterministic choices can be resolved in a way that an accepting run is generated. In the case of the Büchi acceptance condition this means that there is a run of $A$ on $\alpha$ that visits accepting states infinitely often.

**Example 2.8 (Nondeterministic Büchi Automaton)**
Consider the nondeterministic Büchi automaton $A = (Q, q_0, \Sigma, \delta, F)$ in Figure 2.4. This is an automaton over the alphabet $\Sigma = \{a, b\}$. The states from $Q$ are depicted as circles. The initial state $q_0$ is indicated by an incoming arrow and the accepting state $q_1 \in F$ is marked with double circles. The transitions are given by arrows leading from a state to its successor states. The transitions are labeled with the letters from $\Sigma$ that have to be present in the input word in order to enable this transition.

The infinite word $a(b)^\omega$ is in $\mathscr{L}(A)$ because the run $q_0(q_1)^\omega$ generated by this word is accepting since $q_1$ is an accepting state that is visited infinitely often. The word $a(b)^\omega$ also generates the run $(q_0)^\omega$ which is not accepting but nevertheless, $a(b)^\omega \in \mathscr{L}(A)$ because it is enough if one accepting run exists.

The infinite word $b^\omega$ is not in $\mathscr{L}(A)$ because we cannot find an accepting run of $A$ on that word. The only run generated by this word is $(q_0)^\omega$ which is not accepting. □

Nondeterministic Büchi automata are in the following extended to alternating Büchi automata.

**Alternating Büchi Automata**
Alternating Büchi automata do not only allow nondeterministic transitions to a successor state but also universal transitions that lead to several states at the same time. It is then required that the runs through all these states are accepting.

Formally an *alternating Büchi automaton* $A = (Q, q_0, \Sigma, \delta, F)$ consists of the following components:

- $Q$ is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $\Sigma$ is a finite alphabet.
- $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is the transition function.

  $\mathbb{B}^+(Q)$ denotes the set of positive Boolean formulas over the set of states. These formulas are generated by the following grammar for $q \in Q$:

  $$\phi ::= true \mid false \mid q \mid \phi \wedge \phi \mid \phi \vee \phi$$

  The conjunction ($\wedge$) of states represents a universal choice whereas the disjunction ($\vee$) of states represents a nondeterministic choice.
- $F \subseteq Q$ is a set of accepting states.

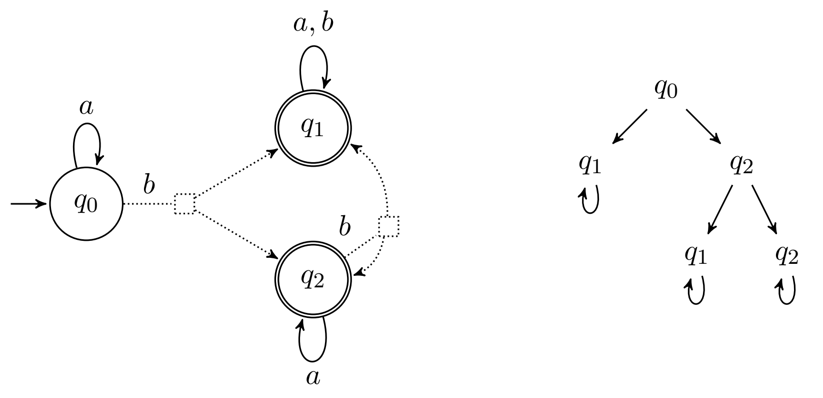


Figure 2.5: Alternating Büchi Automaton $B$ (left) and run of $B$ on word $bb(a)^\omega \in \mathscr{L}(B)$ (right)

Note that a run through an alternating Büchi automaton is an infinite $Q$-labeled tree because one state may have several successor states when taking a universal transition. Such a run tree is accepting if every infinite branch of the tree is accepting. We will not go more into detail but give an intuition for this in the following example.

**Example 2.9 (Alternating Büchi Automaton)**
Consider the alternating Büchi automaton given in Figure 2.5 on the left side. The universal transitions are represented with dotted edges. For example, $\delta(q_0, b)$ returns the positive boolean formula $q_1 \wedge q_2$ requiring that both states accept the rest of the input word.

The run of $B$ on the word $bb(a)^\omega$ is given on the right side of Figure 2.5. The run starts in the initial state of $B$. Because the first input letter is $b$, the universal transition is taken to $q_1$ and $q_2$. In $q_1$ the remaining input $b(a)^\omega$ produces the run $q_1^\omega$. In state $q_2$ the next letter $b$ of the input word leads again to $q_1$ and $q_2$ but now by taking the universal transition starting in state $q_2$. The input of $a$ does not cause any state transitions.

The word $bb(a)^\omega$ is in $\mathscr{L}(B)$ because every branch in the run of $B$ on that word visits an accepting state infinitely often. $\square$

For every LTL formula $\varphi$ there is an equivalent alternating Büchi automaton $A_\varphi$ as presented in [27]. This automaton is constructed by introducing a state for every subformula of $\varphi$ and its negation. The transition function then explicitly represents the temporal requirements of these LTL formulas. The successor of a state corresponding to $\bigcirc\varphi'$ is, for example, the state $\varphi'$.

With a similar construction presented in [15] it is possible to give an equivalent alternating Büchi automaton for every HyperLTL formula $\varphi_H$. For the LTL formula in $\varphi_H$ this construction differs in the alphabet. In [15] the automaton works on tuples of states of a corresponding Kripke structure whereas in [27] it reasons about sets of atomic propositions. After computing the automaton for the LTL subformula, the quantifier prefix is handled by the construction presented in [15].

Although alternating Büchi automata are more concise than nondeterministic Büchi automata, both are equally expressive. This was shown by Miyano and Hayashi in 1984 [18] by giving a construction that translates every alternating Büchi automaton into a nondeterministic Büchi automaton. This construction basically gives a nondeterministic Büchi automaton that accepts all accepting runs (infinite $Q$-labeled trees) of the alternating Büchi automaton.

It directly follows that for every HyperLTL formula there exists a nondeterministic Büchi automaton that formalizes the same language. The opposite direction holds as well. For every nondeterministic Büchi automaton an equivalent HyperLTL formula can be given:

**Construction 2.10 (HyperLTL formula $\varphi_A$)**
Let $A = (Q, q_0, 2^{AP}, \delta, F)$ be a nondeterministic Büchi automaton.

In a first step a QPTL formula $\phi_A$ is calculated from the given Büchi automaton as follows ([13], [26]):

$$\begin{aligned}
\phi_A := \exists\ at_{q_0}. \dots \exists\ at_{q_n}.\ at_{q_0} && (2.1)\\
\wedge\ \Box\,( \bigvee_{(q,A,q')\in\delta} at_q \wedge \bigcirc at_{q'} \wedge (\bigwedge_{p\in A} p) \wedge (\bigwedge_{p\in AP\setminus A} \neg p) && (2.2)\\
\wedge\ \Box\,(\bigwedge_{i=1}^{n} \bigwedge_{j\neq i} \neg(at_{q_i} \wedge at_{q_j})) && (2.3)\\
\wedge\ \Box\,\Diamond \bigvee_{q\in F} at_q && (2.4)
\end{aligned}$$

For every state $q_i \in Q$ a fresh atomic proposition $at_{q_i}$ is introduced by a new propositional quantifier $\exists at_{q_i}$. The behavior of the automaton $A$ is encoded in the QPTL formula $\phi_A$ as it simulates the runs through the automaton using the quantified propositions $at_{q_i}$.

- Every run starts in the initial state $q_0$. (2.1)
- The next state of the run is reached via a transition of the automaton on the currently present atomic propositions. (2.2)
- The automaton is always in only one distinct state. (2.3)

- A run is accepting if it infinitely often visits an accepting state. (2.4)

Thus the QPTL formula $\varphi_A$ accepts every trace that is in the language of the automaton $A$, i.e. $\mathscr{L}(\phi_A) = \mathscr{L}(A)$. Every accepting run of $A$ then uniquely defines the truth values of the atomic propositions $q_i$.

In [9] it is stated that QPTL is subsumed by HyperLTL. So we know that there is a HyperLTL formula $\varphi_A$ that is equivalent to $\phi_A$ and thus has the same language $\mathscr{L}(A)$.

The HyperLTL formula $\varphi_A$ is obtained by applying the following three steps to the QPTL formula $\phi_A$:

1. Rename every proposition $at_{q_i}$ that is bounded by a propositional quantifier with a fresh name $q_i'$.
2. Replace the propositional quantifiers $\exists\, q_i'$ by the trace quantifiers $\exists\, \pi_{q_i'}$ for all variables $q_i'$ introduced in the step before.
3. Replace the atomic propositions $q_i'$ by $(q_i')_{\pi_{q_i'}}$ in the body of the formula.

The resulting formula is the HyperLTL formula $\varphi_A$ with $\mathscr{L}(\varphi_A) = \mathscr{L}(A)$.□

The following example shows how this construction is applied to a given automaton.

**Example 2.11 (Application of Construction 2.10)**
Consider the nondeterministic Büchi automaton $A$ in Figure 2.4.

An equivalent QPTL formula $\phi_A$ is given below:

$$\phi_A = \exists\, at_{q_0}.\ \exists\, at_{q_1}.\ at_{q_0} \tag{2.5}$$
$$\wedge\ \square\,( \bigvee_{(q,a,q')\in\delta} at_q \wedge a \wedge \bigcirc\, at_{q'} \wedge ( \bigwedge_{b\in AP\setminus\{a\}} \neg b)) \tag{2.6}$$
$$\wedge\ \square\,(\neg(at_{q_0} \wedge at_{q_1}) \tag{2.7}$$
$$\wedge\ \square\ \diamond\ at_{q_1} \tag{2.8}$$

(2.5) requires that the run through $A$ starts in the initial state. (2.6) encodes all transitions of the Büchi automaton $A$ by encoding the relation from the states and the present atomic propositions to their successor states. (2.7) formalizes that the automaton is only in a single state at a time and (2.8) requires that the accepting state is visited infinitely often. The QPTL formula $\phi_A$ therefore perfectly simulates the automaton $A$.

Given this QPTL formula $\phi_A$, an equivalent HyperLTL formula $\varphi_A$ is computed following the three steps in Construction 2.10:

$$\begin{aligned}
\varphi_A = \exists\, \pi_{q'_0}.\ \exists\, \pi_{q'_1}.\ (q'_0)_{\pi_{q'_0}} && (2.9)\\
\wedge\ \Box\,(\bigvee_{(q,a,q')\in\delta} q_{\pi_q} \wedge a \wedge \bigcirc\, q'_{\pi_{q'}} \wedge (\bigwedge_{b\in AP\setminus\{a\}} \neg b)) && (2.10)\\
\wedge\ \Box\,(\neg(\,(q'_0)_{\pi_{q'_0}} \wedge (q'_1)_{\pi_{q'_1}}\,)) && (2.11)\\
\wedge\ \Box\,\Diamond\,(q'_1)_{\pi_{q'_1}} && (2.12)
\end{aligned}$$

Theorem 1 in [9] immediately gives us that $\mathscr{L}(\varphi_A) = \mathscr{L}(A)$.

The quantified traces in $\varphi_A$ each represent one automaton state and together they can be interpreted as a run through the automaton $A$. $\square$

In that way we can give an equivalent HyperLTL formula $\varphi_A$ for every nondeterministic Büchi automaton $A$.

## 2.4 Properties of Proof Systems

In the following we will introduce some standard properties of proof systems. A *proof system* in general is a set of proof (or inference) rules where every rule is of the following general form:

$$\frac{\begin{array}{c}\textsc{System'} \models \textsc{Specification'}\\ \textsc{System''} \models \textsc{Specification''}\end{array}}{\textsc{System} \models \textsc{Specification}} \quad \begin{array}{l}\left.\right\}\text{Premises}\\ \left.\right\}\text{Conclusion}\end{array}$$

The rule then allows to conclude that, given all premises hold, the conclusion of the rule holds as well.

A single proof rule is called *sound* if this implication holds. Assuming the premises to be valid it is possible to prove that the conclusion holds as well without using the proof rule that is to be proven sound. A proof system is sound when every of its proof rules is sound. Using a sound proof system, any statement that can be derived using these rules is indeed valid.

On the other hand, a proof system is called *complete* if every valid statement can be derived using the rules of the system. This means that for every valid statement a sequence of rule applications, called derivation, can be found that yield exactly this statement.

Ideally every proof system should be sound and complete. The soundness guarantees that no wrong statement can be proven by using this proof system and the completeness assures that every valid statement can be shown.

However, this is not possible in practice. Gödel's incompleteness theorems from 1931 [16] state that there can be no proof system where all valid assertions are theorems when considering a sufficiently complex language. This implies that achieving completeness is not possible because there will always be a statement that cannot be proven with a given proof system. In 1978 Cook introduced the idea of *relative completeness* [11]. The idea of relative completeness is to assume a very powerful but unrealistic oracle that decides whether these unprovable statements are valid. By using this oracle it is then possible for the proof system to achieve *completeness relative to* the set of statements that have to be decided by the oracle.

Within the scope of this thesis we will develop a proof system for the linear-time logic HyperLTL. This proof system will make use of the sound ATL* proof system [23] that can prove ATL* properties over infinite state systems [23]. Considering the completeness [23] states:

> Completeness of the proof system is relative to validities in the first-order logic, with fixpoints and cpre, of the underlying theory - the same as required for relative completeness for LTL [17] [...].

So the best we can achieve for the proof system for HyperLTL is the same relative completeness.

When saying that a proof system is relative complete to another proof system we mean that the completeness constraints are the same in both proof systems.

# Chapter 3

# The Proof System for HyperLTL

In this chapter we will show how to transform a HyperLTL formula into an ATL* formula and lift the corresponding Kripke structure into a concurrent game structure. With these modifications we can apply the ATL* proof system [23] in order to prove that a HyperLTL formulas holds on a given Kripke structure. The ATL* proof system was introduced by Slanina et al. in 2006. This proof system reduces the task of proving an ATL* formula on a concurrent game structure to the problem of proving several statements in the underlying assertion language by making use of automata-theoretic results.

Given a concurrent game structure $S$ and an ATL* formula of the form $p \to \mathcal{Q}\varphi$ where $p$ is an assertion, $\mathcal{Q} \in \{\langle\!\langle A' \rangle\!\rangle, [\![A']\!]\}$ is a selective path quantifier for some subset $A'$ of the set of all players and $\varphi$ is an ATL* path formula in negation normal form, the proof system decides whether $S \models p \to \mathcal{Q}\varphi$. In doing so the proof system performs four main steps.

- In the first step a rule called "basic state rule" is applied to obtain statements containing only one selective ATL* quantifier followed by an LTL formula. This is done by introducing assertions for every subformula containing a quantifier and then proving that the resulting formula holds on the system as well as that this assertion implies the replaced subformula.
- On the resulting statements of the form $S \models p \to \mathcal{Q}\varphi$ the "basic path rule" is applied. This rule converts the LTL formula $\varphi$ into an assertion and synchronously composes the system $S$ with the automation $A_\varphi$ for $\varphi$.

- In the next step history variables are introduced in order to ensure that a set of memoryless winning strategies exists for the players in $A'$. These variables capture information about events that happened in the past so that this knowledge is always accessible in the current state and the previous states in the history have not to be considered. The "history rule" introduces these variables formally into the system.
- We know that there are memoryless winning strategies for the formulas resulting from the previous step. These formulas are of the form $p \to \mathcal{Q}q$ where $p$ and $q$ are assertions. To these formulas the "assertion rules" are applied that transform this ATL* formula into assertional validities.

The proof rules of the ATL* proof system as well as an example of how they are used can be found in [23]. In [24] the formal proofs for soundness and completeness of this proof system are given. In the following we will show how a Kripke structure $K$ and a HyperLTL formula $\varphi$ for which we want to prove that $K \models \varphi$ have to be modified in order make use of the proof system for ATL*.

## 3.1 Rule Self-Composition

In order to be able to use the proof system for ATL* for proving that a HyperLTL formula holds on a Kripke structure we have to convert the HyperLTL formula into an ATL* formula and the Kripke structure into a corresponding concurrent game structure. The transformation of the system will be achieved by the idea of self-composition [5], a principle that combines several copies of the same system together into a new system. In that way it is possible to observe and compare several executions of the underlying system at the same time.

Depending on how many executions we want to compare at the same time we have to take that many copies of the system into the self composition. We consider every possible combination of states in the single copies to obtain the set of states of the self-composed system. The transitions between these states are combined by the single transitions that are made in the underlying components independently of each other. This technique was used in [25] to reduce safety properties that reason about two traces through a system to safety properties that reason about only one trace in the self-composed system as this system represents two traces at the same time.

When converting a Kripke structure $K$ into a concurrent game structure $S'$ we have to simulate the quantification over traces with the different players in $S'$. A state in $S'$ is a tuple of $h$ states of the Kripke structure $K$ where $h$ is the number of quantified traces in the corresponding HyperLTL formula. Every player in $S'$ controls one of these tuple components and thus generates a path through $K$ encoded in a path through $S'$ when considering only the respective component. Thus every player represents one of the quantified traces and together they generate a path that defines a trace assignment.

The following construction shows how to formally construct a concurrent game structure $S'$ from a given Kripke structure $K$ for a HyperLTL formula $\varphi$ with $h$ quantified traces. The idea is to self-compose $h$ copies of the Kripke structure $K$ and to introduce $h$ players where every player represents one of the quantified traces as described above.

**Construction 3.1 (Self-Composition)**
Let $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ be a Kripke structure and $\mathcal{Q}\pi_1. \dots \mathcal{Q}\pi_h.\psi$ be a HyperLTL formula with $h$ quantified traces where $\mathcal{Q} \in \{\forall, \exists\}$. We construct the concurrent game structure $S' = (A, Q, AP, l, d, \delta)$ as follows:

- $A = \{p_{\pi_g} \mid 1 \leq g \leq h\}$
- $Q = (Q_K)^h$

  where $M^h$ is defined for a set $M$ inductively as follows: $M^1 = M$ and $M^h = M \times M^{h-1}$ for $h > 1$.
- $AP = \{a_{\pi_g} | a \in AP_K, 1 \leq g \leq h\}$
- $l$: $\forall q \in Q.\ l(q) = l((q^{\pi_1}, \dots, q^{\pi_h})) = (l_K(q^{\pi_1}))_{\pi_1} \cup \dots \cup (l_K(q^{\pi_h}))_{\pi_h}$

  where for a set $M$ $M_\pi$ is defined as $\{m_\pi | m \in M\}$. In order to uniquely identify the states components we added a superscript $\pi_g$ to every $q \in Q_K$ indicating the trace quantifier the current state component belongs to.
- $d$: $\forall q \in Q. \forall 1 \leq g \leq h.\ d_{p_{\pi_g}}(q) = d_{p_{\pi_g}}((q^{\pi_1}, \dots, q^{\pi_h})) = d_K(q^{\pi_g})$
- $\delta$: $\forall q \in Q. \forall (i_1, \dots, i_h) \in D(q).\ \delta(q, (i_1, \dots, i_h))$
  $= \delta((q^{\pi_1}, \dots, q^{\pi_h}), (i_1, \dots, i_h))$
  $= (\delta_K(q^{\pi_1}, i_1), \dots, \delta_K(q^{\pi_h}, i_h))$

This system operates on states that are tuples of Kripke structure states. Every component of one such tuple corresponds to one trace quantifier. Let $q = (q^{\pi_1}, \dots, q^{\pi_h})$ be a state in $S'$ and $q^{\pi_g} \in Q_K$ for $1 \leq g \leq h$. The successor state $q'$ is computed by allowing every player to choose the next move in his copy of the Kripke structure $K$, i.e. every move in the move vector corresponds to one component in the state tuple $(q^{\pi_1}, \dots, q^{\pi_h})$. Then the respective successor states are computed independently by applying the transition function $\delta_K$ to the respective state and the corresponding move.

After computing these successor states in $K$ the successor state $q'$ is fully determined. $\square$

When $S'$ is used as a model for an ATL* formula then the moves in the move vectors are chosen by the strategies of the players. As we will see the players corresponding to an existential trace quantifier have to choose their moves first. This means that the successor states for these components are already fixed when the universal players have to choose their moves and thus the universal players may adapt their choices depending on the moves the existential players chose. As soon as all moves are fixed, a unique successor state is identified.

Every path $u = q_0\, q_1\, q_2 \dots$ through the concurrent game structure $S'$ defines a unique trace assignment $\Pi_u$. In order to see this we first have to define the projection:

**Definition 3.2 (Projection $pr_g$)**
Let $q = (q_1, \dots, q_h)$ be in $Q^h$, i.e. $q$ is a tuple containing $h$ elements from the set $Q$ as its components. Then $pr_g : Q^h \to Q$ is the *$g$-th projection* returning the component of $q$ at position $g$:

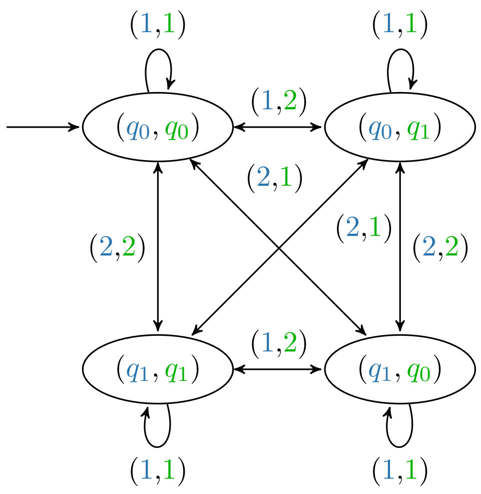
$$\forall\ 1 \leq g \leq h.\ pr_g(q) = q_g \qquad \square$$

Using the $g$-th projection on every state $q_t$ of a path $u = q_0\, q_1\, q_2 \dots$ through $S'$ we obtain the path $u_g = pr_g(\,u[0])\ pr_g(\,u[1])\ pr_g(\,u[2]) \dots$ through the Kripke structure $K$ that is encoded in $u$ at the $g$-th component. To get the trace generated by the path $u_g$ the labeling function $l_K$ has to be applied to every state of the path. Like that we can access the trace corresponding to every player. We thus can compute a unique trace assignment $\Pi_u$ from a path $u$ through $S'$ by using the projections $pr_1$ to $pr_h$ and the labeling function $l_K$ of the underlying Kripke structure:

**Definition 3.3 (Trace assignment $\Pi_u$)**
Let $u$ be a path through a concurrent game structure $S'$. $S'$ was gained by applying Construction 3.1 to a Kripke structure $K$ with a labeling function $l_K$ for a formula with $h$ trace quantifiers. The trace assignment $\Pi_u$ is defined as follows:

$$\forall\ 1 \leq g \leq h.\ \Pi_u(\pi_g) = l_K(\ pr_g(\ u[0]\ ))\ l_K(pr_g(u[1]))\ l_K(pr_g(u[2])) \dots \qquad \square$$

Note that the labeling function $l$ defined in Construction 3.1 does not simply apply the labeling function of the Kripke structure $l_K$ to every component. If this was the case $l'$ would return a tuple of sets of atomic propositions, i.e. $l'((q^{\pi_1}, \dots, q^{\pi_h})) = (l_K(q^{\pi_1}), \dots, l_K(q^{\pi_h}))$. Instead the labeling function $l$ returns the union of all these sets where the elements in the sets are indexed with a unique name indicating the tuple component they would occur in. These indexes allow to reconstruct the sets generated by the single Kripke



with $A = \{p_{\pi_1}, p_{\pi_2}\}$.

Figure 3.1: Concurrent game structure $S'$ gained by Construction 3.1 from Kripke structure $K$ (Figure 2.1) for a HyperLTL formula with two trace quantifiers

structure states by considering all atomic propositions in $l(q)$ having the desired index. $(l_K(q^{\pi_g}))_{\pi_g} = \{a_{\pi_g} | a_{\pi_g} \in l(q)\}$

Construction 3.1 is applied to the Kripke structure $K$ given in Figure 2.1 in the following example.

**Example 3.4 (Application of Construction 3.1)**
Reconsider the Kripke structure $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ from Figure 2.1 and the HyperLTL formula $\forall\pi_1.\exists\pi_2.\square(a_{\pi_1} \leftrightarrow a_{\pi_2})$ from Example 2.7. By applying Construction 3.1 we obtain the concurrent game structure $S' = (A, Q, AP, l, d, \delta)$ depicted in Figure 3.1.

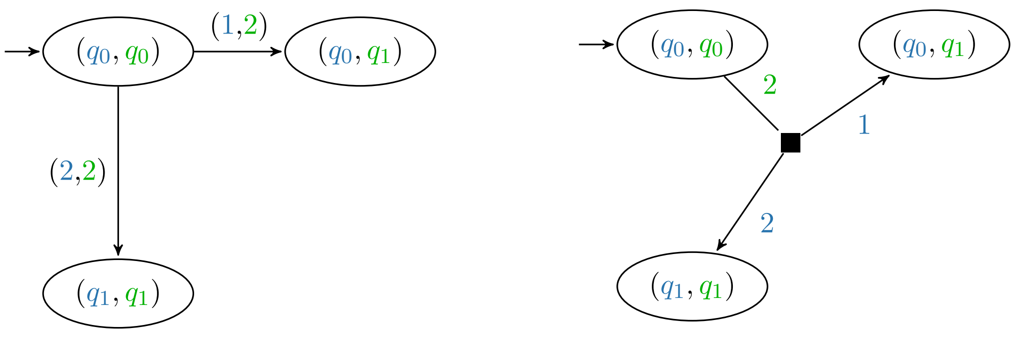


Figure 3.2: Order of move selection in the concurrent game structure $S'$

The states of $S'$ are pairs of states of $K$ for every combination of two $K$-states. To get the set of atomic propositions $AP$ every atomic proposition from $AP_K$ is indexed with every trace variable $\pi_g$ and the result is then interpreted as a fresh set of atomic propositions for $S'$. The labeling function $l$ returns for a state $s = (q_x, q_y)$ for $x, y \in \{0, 1\}$ calculates the labels of the single states $q_x$ and $q_y$ in $K$ and adds the index $\pi_1$ to the elements in $l(q_x)$ and the index $\pi_2$ to $l(q_y)$. Then the union of the resulting sets is returned.

Because the HyperLTL formula contains two trace quantifiers, the concurrent game structure $S'$ will have two players: $A = \{p_{\pi_1}, p_{\pi_2}\}$. The first player represents the universally quantified component whereas the second player generates the traces for the existentially quantified component. The number of possible moves for player $p_{\pi_g}$ with $g \in \{1, 2\}$ in a state $s = (q^{\pi_1}, q^{\pi_2})$ is exactly the number of possible moves from state $q^{\pi_g}$ in the Kripke structure $K$. The single players thus follow the transition function $\delta_K$ within their component independently of the state components the other player controls. The transition function $\delta$ in a state $q$ is defined on the move vectors $D(q)$ that contain one move per player. In the Figure 3.1 the moves chosen by player $p_{\pi_1}$ are given in blue, the moves of player $p_{\pi_2}$ in green.

This concurrent game structure will be used as the model for an ATL* formula so it is important to recall that the moves of the existential player $p_{\pi_2}$ have to be fixed before the universal player $p_{\pi_1}$ chooses her moves. We can thus think of the transitions to be divided into two steps as described above. Intuitively the transitions on the left side of Figure 3.2 can be viewed as illustrated in Figure 3.2 on the right side. □

We know how to obtain the corresponding concurrent game structure. It is left to transform the HyperLTL formula into an ATL* formula. Assume we have $m$ universal and $n$ existential quantifiers in the quantifier prefix of a HyperLTL formula, i.e. the formula is of the form $\forall\pi_1. \dots \forall\pi_m.\exists\pi'_1. \dots \exists\pi'_n.\psi$ where $\psi$ is a quantifier-free LTL formula. The universal quantified traces $\pi_i$ can be viewed as chosen arbitrarily whereas the existential quantified traces $\pi'_i$ are chosen adaptively to the choice of the universal quantified traces. Thus in $S'$ we require the players $p_{\pi'_i}$ to have a strategy to choose the corresponding traces which means that these players will be the ones selected by the ATL* quantifier resulting in $\langle\langle p_{\pi'_1}, \dots, p_{\pi'_n}\rangle\rangle\psi$. In order to meet the syntactical requirements of the ATL* proof system we wrap this formula as the conclusion into an implication with *true* as the premise. Putting all this together we obtain the inference rule displayed in Figure 3.3.

**Theorem 3.5 (Soundness of Rule Self-Composition)**
The inference rule Self-Composition in Figure 3.3 is sound. □

$$\frac{S' \models \mathit{true} \to \langle\!\langle p_{\pi'_1}, \dots, p_{\pi'_n} \rangle\!\rangle \psi}{K \models \forall \pi_1 \dots \forall \pi_m . \exists \pi'_1 \dots \exists \pi'_n .\ \psi}$$

where $S'$ is built from $K$ by using Construction 3.1.

Figure 3.3: Rule Self-Composition

**Proof** We prove this claim by contradiction.

Assume $S' \models \mathit{true} \to \langle\!\langle p_{\pi'_1}, \dots, p_{\pi'_n} \rangle\!\rangle \psi$ and $K \not\models \forall \pi_1 \dots \forall \pi_m . \exists \pi'_1 \dots \exists \pi'_n .\ \psi$. $S' \models \mathit{true} \to \langle\!\langle p_{\pi'_1}, \dots, p_{\pi'_n} \rangle\!\rangle \psi$ implies that $S' \models \langle\!\langle p_{\pi'_1}, \dots, p_{\pi'_n} \rangle\!\rangle \psi$ holds and thus the players $p_{\pi'_k}$ together have a set of strategies $F_{\{p_{\pi'_1}, \dots, p_{\pi'_n}\}}$, one strategy $\sigma_{p_{\pi'_k}}$ for every player $p_{\pi'_k}$ in $S'$ with $1 \leq k \leq n$ so that they together always can ensure that the formula $\psi$ is fulfilled.
*Idea:* We now use these strategies to construct paths on the Kripke structure $K$ such that the traces of these paths guarantee to fulfill $\psi$ together with an arbitrary choice of the universal quantified traces. This will then contradict to the assumption that the formula $\forall \pi_1 \dots \forall \pi_m . \exists \pi'_1 \dots \exists \pi'_n .\ \psi$ does not hold on $K$.

Let every strategy $\sigma_{p_{\pi'_k}}$ in $F_{\{p_{\pi'_1}, \dots, p_{\pi'_n}\}}$ on a finite history $q_0 q_1 \dots q_t$ with every state $q_h = (s_h^{\pi_1}, \dots, s_h^{\pi_m}, s_h^{\pi'_1}, \dots, s_h^{\pi'_n})$ for $0 \leq h \leq t$ be defined as follows:
$\forall t \in \mathbb{N}.\ \sigma_{p_{\pi'_k}}(q_0 q_1 \dots q_t) = i$ giving a move $1 \leq i \leq d_K(s_t^{\pi'_k})$ for player $p_{\pi'_k}$ that defines a unique successor state $s_{t+1}^{\pi'_k}$.
When all the players $p_{\pi'_k}$ apply their respective strategy $\sigma_{p_{\pi'_k}}$ from the set $F_{\{p_{\pi'_1}, \dots, p_{\pi'_n}\}}$ in every step we get a path $u = (s_0^{\pi_1}, \dots, s_0^{\pi_m}, s_0^{\pi'_1}, \dots, s_0^{\pi'_n})$ $(s_1^{\pi_1}, \dots, s_1^{\pi_m}, s_1^{\pi'_1}, \dots, s_1^{\pi'_n}) \dots$ in $S'$ fulfilling $\psi$ regardless of which moves were chosen by the players $p_{\pi_1}, \dots, p_{\pi_m}$.

Now consider the second assumption $K \not\models \forall \pi_1 \dots \forall \pi_m . \exists \pi'_1 \dots \exists \pi'_n .\ \psi$. This is equivalent to $K \models \exists \pi_1 \dots \exists \pi_m . \forall \pi'_1 \dots \forall \pi'_n .\ \neg\psi$. Thus we can conclude that in the Kripke structure $K$ there have to be $m$ paths $u_1$ to $u_m$ such that for all possible choices of paths $u'_1$ to $u'_n$ the formula $\psi$ is not fulfilled on the respective traces. Let $u_l = s_{l,0} s_{l,1} s_{l,2} \dots$ for $1 \leq l \leq m$ be exactly these paths. We construct the paths $u'_k$ as follows:
The paths all start with the initial state $q_0$ of the system $K$ and all subsequent states are chosen according to the strategies in $F_{\{p_{\pi'_1}, \dots, p_{\pi'_n}\}}$. Thus for every path $u'_k = s'_{k,0} s'_{k,1} s'_{k,2} \dots$ the following holds: $s'_{k,0} = q_0$ and $s'_{k,t+1} = \delta(s'_{k,t},\ \sigma_{p_{\pi'_k}}((s_{1,0}, \dots, s_{m,0}, s'_{1,0}, \dots, s'_{n,0}) \dots (s_{1,t}, \dots, s_{m,t}, s'_{1,t}, \dots, s'_{n,t})))$ for

$t \in \mathbb{N}$ and $\sigma_{p_{\pi'_k}}$ being the strategy from player $p_{\pi'_k}$ in the concurrent game structure $S'$.
So we found a set of paths $u'_k$ that together with the given paths $u_l$ the respective traces fulfill the formula $\psi$ in the system $K$. This is a contradiction to the assumption that $K \not\models \forall\pi_1. \dots \forall\pi_m.\exists\pi'_1. \dots \exists\pi'_n.\ \psi$. Thus we can conclude that the rule Self-Composition is sound. ■

After the application of the rule Self-Composition (Figure 3.3) we use the ATL* proof system presented at the beginning of the chapter to prove $S' \models true \to \langle\langle p_{\pi'_1}, \dots, p_{\pi'_n}\rangle\rangle\psi$. Because of the soundness of this rule we know that if $S' \models true \to \langle\langle p_{\pi'_1}, \dots, p_{\pi'_n}\rangle\rangle\psi$ holds then $K \models \forall\pi_1. \dots \forall\pi_m.\exists\pi'_1. \dots \exists\pi'_n.\ \psi$ holds too. Together with the soundness of the ATL* proof system we know that the resulting proof system consisting of Rule 3.3 and the ATL* proof rules is sound. That means every statement that can be proven is indeed correct.

Note that the ATL* proof system will skip its first step (applying the "basic state rule") and directly start with step two. This is possible because the ATL* formula gained by Rule Self-Composition in Figure 3.3 already meets the requirements of the second step as it only consists of one ATL* quantifier followed by an LTL formula. This is the case since we started from a HyperLTL formula having a quantifier prefix followed by an LTL formula that we did not change. The quantifier prefix was transformed into a single selective quantifier by the self-composition rule.

## 3.2 Lack of Completeness

Another desirable property of proof systems in general is completeness. This requires that every correct statement can be proven using the rules of the proof system. Unfortunately this proof system is not yet complete. To see this consider the following counterexample.

**Example 3.6 (Completeness Counterexample)**
Consider again the Kripke structure $K$ from Figure 2.1 together with the HyperLTL formula $\varphi = \forall\pi_1.\exists\pi_2.\Box(a_{\pi_1} \leftrightarrow a_{\pi_2})$. We now want to prove that $\varphi$ holds on $K$ which is the case as explained in Example 2.7. So we start with the correct statement $K \models \forall\pi_1.\exists\pi_2.\Box(a_{\pi_1} \leftrightarrow a_{\pi_2})$ and want to prove it formally using our proof system.

**Proof** We start by applying the Rule Self-Composition in Figure 3.3. In order to do so we construct the concurrent game structure $S'$ as defined in Construction 3.1 (Figure 3.1). Then we obtain the statement $S' \models ture \to \langle\langle p_{\pi_2}\rangle\rangle\Box(a_{\pi_1} \leftrightarrow a_{\pi_2})$ that is left to be proven.

It has to be shown that player $p_{\pi_2}$ has a strategy that ensures that she always chooses to go to the same state as player $p_{\pi_1}$ does. Recall that in ATL* the players that occur in the quantifier have to choose their moves first. That means that $p_{\pi_2}$ has to decide for a move before $p_{\pi_1}$ does so and it is not possible for $p_{\pi_2}$ to simply copy the move of $p_{\pi_1}$. Because of this order there can be no winning strategy for player $p_{\pi_2}$.

To see this assume there was a strategy $\sigma$ for $p_{\pi_2}$ that guarantees to be winning. Then for every finite history $\sigma$ yields a unique move $i$ that player $p_{\pi_2}$ will take so that $p_{\pi_2}$ is then either in a state where $a$ holds or not. In the moment when player $p_{\pi_1}$ has to choose her move the choice of $p_{\pi_2}$ is already made and observable for $p_{\pi_1}$. So $p_{\pi_1}$ can simply choose the move that ends up in exactly the other state ensuring that $a_{\pi_1}$ and $a_{\pi_2}$ have different truth values and thus are not equivalent.

This is a contradiction to the assumption that the strategy $\sigma$ is winning for $p_{\pi_2}$ and we can conclude that no such strategy can exist. ■

Although the HyperLTL formula holds on $K$ we are not able to give a formal proof using this proof system. This means that our proof system is not complete. □

The reason why the proof system is not complete is the finite knowledge that is accessible by strategies on concurrent game structures. A strategy works on the finite information provided through the history that it receives as an argument. So no information about the future moves of any player are known. On the contrary in HyperLTL the whole traces with all their information are accessible when deciding if a formula holds on a given trace assignment.

To obtain a complete proof system we thus need a method to prevent that the information about the future needed to give a winning strategy in the concurrent game structure is lost in the transition from HyperLTL to ATL*. We want to provide all information needed to give the correct strategies if the initial statement holds. In the following chapter we will do exactly this by introducing so called prophecy variables [1].

# Chapter 4

# Completeness of the Proof System

Only a complete proof system guarantees that every correct statement can actually be proven using this system. The reason why the proof system presented in this thesis is not complete yet is that the strategies used in the ATL* semantics can only reason about the history of the path through the system whereas the trace assignment of HyperLTL formulas allows to access the entire traces. In the transition from a HyperLTL formula to an ATL* formula important information is lost that is needed to decide whether the statement that is to be proven holds.

Assume we want to reason about a HyperLTL formula $\mathcal{Q}\pi_1 \ldots . \mathcal{Q}\pi_n.\psi$ with $n$ quantified traces where $\psi$ is a quantifier-free LTL formula and $\mathcal{Q} \in \{\forall, \exists\}$. We will then first transform this formula into an equivalent HyperLTL formula that formalizes the additional information about the future explicitly before we then apply the self-composition in order to use the ATL* proof system. This information is made explicit by introducing so called prophecy variables [1] to the HyperLTL formula that guess if the universally quantified traces are going to behave in a certain way described by a HyperLTL formula. The prophecy variable $p$ represents the guess if this HyperLTL formula $\exists\pi_{\varphi_1} \ldots . \exists\pi_{\varphi_m}.\varphi$ will hold for the universally quantified traces or equivalently $p$ guesses if the universally quantified trace will be in the language of this HyperLTL formula. Assuming this guess to be correct it is then possible to give an adequate strategy for the explicitly quantified players in the concurrent game structure after the application of the self-composition using this knowledge.

However, if the guess is wrong then this still does not affect the correctness because the prophecy variable $p$ is introduced as the premise of an implication where the original formula $\psi$ is the conclusion. Intuitively we gain a HyperLTL formula of the form $\mathcal{Q}\pi_1 \ldots . \mathcal{Q}\pi_n . \exists\pi_{\varphi_1} \ldots . \exists\pi_{\varphi_m} . \square(p \leftrightarrow \varphi) \rightarrow \psi$. If the truth values of the prophecy variable are guessed wrong then the premise is *false* and we thus can conclude anything. But if the guess was correct then the premise is *true* and using this information we can give a strategy guaranteeing that the conclusion will hold too.

In the following we want to capture this idea more formal. While doing so we will restrict our attention to finite state systems and safety properties that are properties requiring that something bad never happens [2].

As a first step we have to identify the pieces of information that the prophecy variables should guess. That means we have to find some formulas $\varphi$ that represent the information needed in order to give a set of winning strategies. Then the prophecy variables guess whether these formulas will hold or not.

In general we want to gain information about how the universally quantified traces will behave in the future so that the existentially quantified traces can be chosen based on this knowledge and thus it is possible to give the strategy. The formulas $\varphi$ may therefore only talk about the future of the universally quantified traces. We will construct these formulas in a way that they describe the language of allowed future moves for the players representing the universally quantified moves in the self-composed system. After that the prophecy variables may guess if the universally quantified traces will meet these requirements. The languages are computed from the self-composed system together with the Büchi automaton for the LTL formula $\psi$. This automaton is built with the construction from [15] so that the alphabet of the automaton is a tuple of states. From the system gained by considering the self-composition together with this automaton these languages are then computed and one formula $\varphi_s$ and one prophecy variable $p_s$ for every state $s$ in this system is introduced.

Because we will consider this modified version of the self-composed system obtained by the Rule Self-Composition (see Figure 3.3) where the automaton for $\psi$ is included we will present a modified rule for the self-composition before we actually see how to introduce the required prophecy variables.

## 4.1 Modifying Rule Self-Composition

The idea is that we will build the product construction of the self-composed system and the nondeterministic Büchi automaton for the formula $\psi$. This construction combines the states of both system by applying the cross product to the two sets of states. Like that the current state of the automaton can be observed without influencing the behavior of the self-composed system.

**Construction 4.1 (Product of Self-Composition and Automaton)** Let $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ be a Kripke structure and $\mathcal{Q}\pi_1.\dots\mathcal{Q}\pi_n.\varphi$ be a HyperLTL formula with $n$ quantified traces where $\mathcal{Q} \in \{\forall, \exists\}$. Construct the concurrent game structure $S' = (A_{S'}, Q_{S'}, AP_{S'}, l_{S'}, d_{S'}, \delta_{S'})$ as described in Construction 3.1. Every instance of the system $K$ in $S'$ describes to one quantified trace in the given HyperLTL formula.

Then construct the alternating Büchi automaton $A = (Q_A, q_{A,0}, \Sigma_A, \delta_A, F_A)$ for the LTL formula $\varphi$ by using the construction as described in [15]. This construction uses tuples of states from $Q_K$ as the alphabet, for every quantified trace in the HyperLTL formula we have one element in the tuples: $\Sigma_A = (Q_K)^n$. A run through the automaton $A$ then is an element of $(Q_K)^n$ and represents exactly $n$ paths where e.g. the first path is recovered by considering only the first component of every element in the run. If we consider the corresponding traces to these paths and interpret the formula $\varphi$ on the trace assignment $\Pi$ assigning the first trace variable to the trace generated by the first path encoded in the run then we see that the run in $A$ is accepting whenever $\Pi \models_K \varphi$.

Convert the alternating Büchi automaton $A$ for $\varphi$ into an equivalent nondeterministic Büchi automaton $A' = (Q_{A'}, q_{A',0}, \Sigma_A, \delta_{A'}, F_{A'})$ with $\mathscr{L}(A) = \mathscr{L}(A')$ by using the standard construction of Miyano and Hayashi [18].

We now construct the concurrent game structure $S = (A, Q, AP, l, d, \delta)$ that captures the different paths through $K$ and the corresponding state of the automaton $A'$ for $\varphi$ as follows:

- $A = A_{S'}$
- $Q = Q_{S'} \times Q_{A'}$
- $AP = AP_{S'}$
- $l$: $\forall q \in Q. l(q) = l((q_{S'}, q_A)) = l_{S'}(q_{S'})$
- $d$: $\forall q \in Q. \forall 1 \leq m \leq n. d_{\pi_m}(q) = d_{\pi_m}((q_{S'}, q_A)) = d_{S'}(q_{S'})$
- $\delta$: $\forall q \in Q. \forall (i_1, \dots, i_n) \in D(q).\ \delta(q, (i_1, \dots, i_n))$
  $= \delta((q_{S'}, q_{A'}), (i_1, \dots, i_n)) = \delta(((q^{\pi_1}, \dots, q^{\pi_n}), q_{A'}), (i_1, \dots, i_n))$
  $\in \{((\delta_K(q^{\pi_1}, i_1), \dots, \delta_K(q^{\pi_n}, i_n)), q'_{A'}) \mid$
  $q'_{A'} \in \delta_{A'}(q_{A'}, (\delta_K(q^{\pi_1}, i_1), \dots, \delta_K(q^{\pi_n}, i_n)))\}$

Note that this concurrent game structure is nondeterministic because the nondeterministic automaton $A'$ is added. The transition function $\delta$ has become a transition relation and does not return a unique state but a set of states that differ only in their automaton component $q'$. Assume the system $S$ is in a state $(s, q)$ where $s$ is a state in $S'$ and $q$ is a state in $A'$. The successor state $(s', q')$ is then computed intuitively as follows:

- The successor state $s'$ of state $s$ in the self-composed system $S'$ is computed. Recall that this is a concurrent game structure used as a model for an ATL* formula and therefore the explicitly quantified players have to choose their moves first.
  - The players representing an existentially quantified trace choose their moves.
  - Then all other players choose their moves.
  - Together all these moves form a complete move vector $i \in D_{S'}(s)$ and the transition function then yields a unique successor state $s' = \delta_{S'}(s, i)$.
- Then the successor state $q'$ of state $q$ in the automaton $A'$ is chosen according to the state $s'$ computed in the previous step.
  - Calculate the set of possible successor states $\delta_{A'}(q, s')$
  - Choose a state $q'$ from this set nondeterministically.
- Combining these results we receive the successor state $(s', q')$.

We call a path $u$ through $S$ accepting if the run $r$ through $A'$ encoded in the automaton component of this path is accepting. Using the projection we can compute $r = r_0 r_1 r_2 \dots$ from $u = u_0 u_1 u_2 \dots$ where $r_i$ is a state in the automaton $A'$ and $u_i = (s_i, q_i)$ is a state in $S$ for $i \in \mathbb{N}$: $r_i = pr_2(u_i)$

$S$ can be interpreted directly as the nondeterministic Büchi automaton $S = (Q_S, q_{S,0}, \Sigma_S, \delta_S, F_S)$ where $q_{S,0} = (q_{S',0},\ q_{A',0})$ is the start state with $q_{S',0} \in Q_{S'}$ being the state that has the start state of the Kripke structure $q_0 \in Q_K$ at every tuple component. The set of accepting states is determined by the accepting states of $A'$: $F_S = \{(q_{S'}, q_{A'}) | q_{S'} \in Q_{S'}, q_{A'} \in F_{A'}\}$. The alphabet of the automaton then is the set of all move vectors in $S$: $\Sigma_S = (\mathbb{N}^*)^n$. The transition relation is $\delta_S : Q_S \times (\mathbb{N}^*)^n \to 2^{Q_S}$ and for every state $q \in Q_S$ and every move vector $I \in [\mathbb{N}^*]^n$ $\delta_S(q, I)$ is only defined if $I \in D(q)$.

By adding $A'$ to $S'$ the current automaton state is directly accessible in the states of $S$ and we can use this information in order to compute the strategies required in the ATL* setting as we will see below. □

**Example 4.2 (Application of Construction 4.1)**
Again we consider the Kripke structure K in Figure 2.1 and the formula $\forall \pi_1 . \exists \pi_2 . \Box(a_{\pi_1} \leftrightarrow a_{\pi_2})$ from Example 2.7.

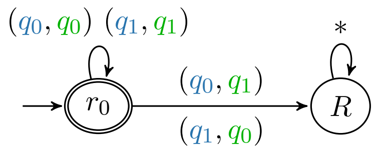


Figure 4.1: Büchi automaton $A'$

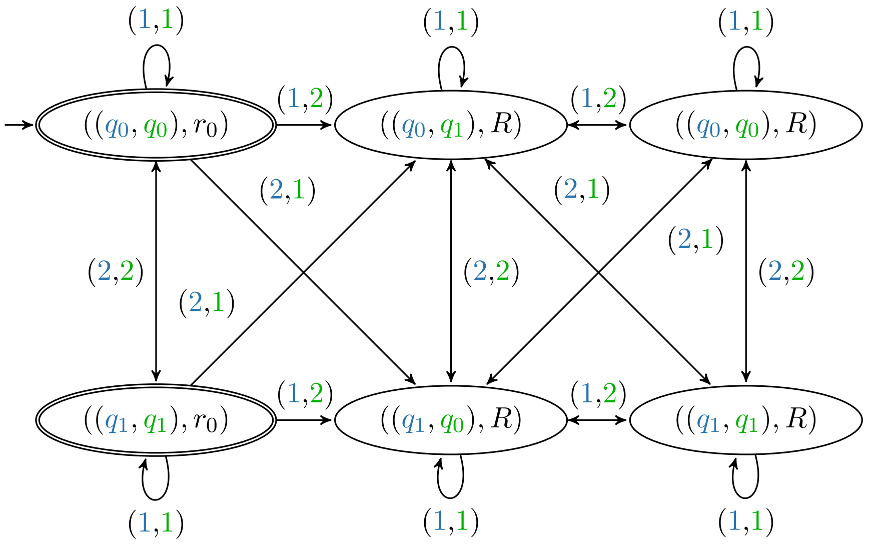


Figure 4.2: System $S$ gained by Construction 4.1 by combining $S'$ and $A'$

$$\frac{S \models \mathit{true} \rightarrow \langle\!\langle p_{\pi'_1}, \ldots, p_{\pi'_n} \rangle\!\rangle \varphi}{K \models \forall \pi_1 . \ldots \forall \pi_m . \exists \pi'_1 . \ldots \exists \pi'_n . \, \varphi}$$

where $S$ is built from $K$ by using Construction 4.1.

Figure 4.3: Modified Rule Self-Composition

Construction 4.1 calculates the concurrent game structure $S'$ (see Figure 3.1). Moreover, a nondeterministic Büchi automaton $A'$ for $\square(a_{\pi_1} \leftrightarrow a_{\pi_2})$ is built. The automaton in Figure 4.1 is even a deterministic Büchi automaton that accepts all paths through $S'$ whose traces fulfill $\square(a_{\pi_1} \leftrightarrow a_{\pi_2})$. The alphabet $\Sigma$ of this automaton consists of pairs of states in $K$ because we have two trace quantifiers in the HyperLTL formula.

The concurrent game structure $S$ is then gained by combining $S'$ and $A'$ using Construction 4.1. The resulting system $S$ is depicted in Figure 4.2. The two states $(q_0, q_1, r_0)$ and $(q_1, q_0, r_0)$ are unreachable and therefore not given in Figure 4.2.

The states of $S$ are pairs of states of $S'$ and states of $A'$. Every state in $Q$ is labeled with the same atomic propositions that its $S'$ part is labeled with. The transition function $\delta$ computes the transitions in $S'$ and in $A'$ and updates both state components accordingly. In this example the resulting concurrent game structure $S$ is deterministic because the Büchi automaton $A'$ is. The states containing an accepting state from $A'$ are marked with double circles such that accepting paths through $S$ can be identified.

The set of players $A$ is the same as in $S'$ and the moves chosen by player $p_{\pi_1}$ are given in blue and the moves chosen by player $p_{\pi_2}$ are green. □

We require that the system $S$ used in rule Self-Composition is generated by using this construction above such that we gain the modified rule for the self-composition presented in Figure 4.3. This modification does not affect the soundness of the rule Self-Composition at all.

**Theorem 4.3 (Soundness of Modified Rule Self-Composition)**
The modified inference rule Self-Composition in Figure 4.3 is sound. □

**Proof** The proof of this claim proceeds mostly analogous to the proof of Theorem 3.5. The main difference is that the system $S$ now is nondeterministic because every state has the additional component representing a state from the nondeterministic automaton.

Then from the assumption that $S \models \mathit{true} \rightarrow \langle\!\langle p_{\pi'_1}, \ldots, p_{\pi'_n} \rangle\!\rangle \varphi$ holds we can conclude that there is a winning set of strategies $F_{\{p_{\pi'_1}, \ldots, p_{\pi'_n}\}}$. In the case of a nondeterministic concurrent game structure a set of strategies is called winning if given an arbitrary set of strategies for the other players there is always a path through $S$ such that $\varphi$ holds on that path. If all strategies are fixed the paths that are generated when applying these strategies only differ in the nondeterministic choice of the automaton $A'$. The component that represents a path through $S'$ is the same in all these paths. The assumption implies that this path is accepted by $A'$. This means that there is an accepting run of $A'$ on this path. The path generated by the fixed set of strategies that encodes exactly this run through $A'$ is then an accepting path in $S$.

We use the strategies in $F_{\{p_{\pi'_1},\dots,p_{\pi'_n}\}}$ to construct $n$ paths $u'_k = s'_{k,0}s'_{k,1}s'_{k,2}\dots$ with $1 \leq k \leq n$ in order to contradict to the assumption that $K \models \exists\pi_1.\dots\exists\pi_m.\forall\pi'_1.\dots\forall\pi'_n.\ \neg\varphi$.

Let $u_l = s_{l,0}s_{l,1}s_{l,2}\dots$ for $1 \leq l \leq m$ be paths that generate the existentially quantified traces. We construct the paths $u'_k$ in exactly the same way as in the proof of Theorem 3.5 by using the strategies $\sigma_{p_{\pi'_k}}$ from the set $F_{\{p_{\pi'_1},\dots,p_{\pi'_n}\}}$ and the paths $u_l$.

Together the paths $u_l$ and $u'_k$ define a path $s = s_0s_1s_2\dots$ with every state being of the form $s_t = (s_{1,t},\dots,s_{m,t},s'_{1,t},\dots,s'_{n,t})$ for all $t \in \mathbb{N}$. This sequence of states can be interpreted as input to the automaton $A'$ which then generates one run $r = q_0q_1q_2\dots$ that is chosen nondeterministically from the set of all runs on this input. We then combine $s$ and $r$ and obtain a path $(s_0,q_0)(s_1,q_1)(s_2,q_2)\dots$ through the system $S$. By our assumption we know that if $r$ was chosen correctly, i.e. $r$ is an accepting run in $A'$ then the strategies in $F_{\{p_{\pi'_1},\dots,p_{\pi'_n}\}}$ guarantee that $\varphi$ will be fulfilled by the generated traces. Then it immediately follows that these traces wrapped in the respective trace assignment $\Pi$ fulfill $\varphi$ on the Kripke structure $K$: $\Pi \models_K \varphi$. Even if a non-accepting run $r$ was selected we can conclude that the traces generated by the paths $u'_k$ satisfy $\varphi$ because the choice of the automaton run has no influence on the truth value of the formula $\varphi$ under a given trace assignment.

Because we always can find the adequate paths $u'_k$ for all paths $u_l$ such that the generated traces satisfy $\varphi$ we found a contradiction to the assumption that $K \models \exists\pi_1.\dots\exists\pi_m.\forall\pi'_1.\dots\forall\pi'_n.\ \neg\varphi$ and we can conclude that $K \models \forall\pi_1.\dots\forall\pi_m.\exists\pi'_1.\dots\exists\pi'_n.\ \varphi$ and the modified Rule 4.3 is sound. ■

By using this modified rule we then can reason about both, the states of the self-composed system $S'$ and the states of the automaton $A'$. Therefore we are able to access all information needed in the ATL* setting in order to ensure that there is a winning set of strategies for the players representing the existentially quantified traces. This information has to be made explicit in order to give the actual strategies of these players.

## 4.2 Introducing Prophecy Variables

In order to show how to compute a winning set of strategies for the existential players we first have to ensure that the prophecy variables $p_s$ and the pieces of information they guess are chosen adequately. As described above this information is represented by HyperLTL formulas $\varphi_s$ that describe the allowed future behavior of the universally quantified traces starting from every state $s$. We want to see how these formulas are constructed formally. After that we will see how the prophecy variables are introduced into the formula by the corresponding proof rule.

**Construction 4.4 (Computation of $\varphi_s$)**
Let a finite state Kripke structure $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ and a HyperLTL formula of the form $\forall\pi_1. \ldots \forall\pi_m. \exists\pi'_1. \ldots \exists\pi'_n.\ \psi$ be given.

**Step 1: Calculate $S$**
Construct the concurrent game structure $S = (A, Q, AP, l, d, \delta)$ from $K$ and the formula as described in Construction 4.1. The states of this system are pairs of a tuple of $m + n$ Kripke structure states and a state of the nondeterministic Büchi automaton $A'$ for the LTL formula $\psi$. There is one player for every quantified trace from the original HyperLTL formula in the set $A$. We have $m$ universal players $p_{\pi_l}$ for $1 \leq l \leq m$ and $n$ existential players $p_{\pi'_k}$ for $1 \leq k \leq n$. The transitions are the combination of the single moves in the $K$ components such that $S$ reasons about infinite sequences of move vectors.

**Step 2: Existential Projection to $S_\exists$**
As mentioned above the formulas $\varphi_s$ that we want to construct may only reason about the universally quantified traces. However, the language of system $S$ captures both the moves of the players representing the universally and the existentially quantified traces. This additional information is removed by applying the existential projection to $S$ resulting in a system $S_\exists$ which projects away the moves for the existential players. The system $S_\exists$ then accepts all move vector sequences for the universal players for which there exist a corresponding move vector sequence for the existential players such that the resulting path $u$ through the system is accepting and equivalently the trace assignment $\Pi_u$ is a model for the initial HyperLTL formula.

Formally the system $S_\exists = (A_{S_\exists}, Q, AP, l, d, \delta_{S_\exists})$ is gained by applying the existential projection as follows:

- $A_{S_\exists} = \{p_{\pi_l} | 1 \leq l \leq m\}$
- $\delta_{S_\exists} = \{(q, J, q') | \exists I \in (\mathbb{N}^*)^n . (q, J \circ I, q') \in \delta, q, q' \in Q, J \in (\mathbb{N}^*)^m\}$ where $\circ$ is defined on two tuples $J = (j_1, \ldots, j_m)$ and $I = (i_1, \ldots, i_n)$ as follows: $J \circ I = (j_1, \ldots, j_m, i_1, \ldots, i_n)$
  $J \circ I \in (\mathbb{N}^*)^{m+n}$ is a move vector in $S$ and the existential projection transforms this into a move vector in $S_\exists$ by deleting the moves for the existential players.

**Step 3: HyperLTL formulas $\exists\pi_s^1. \ldots \exists\pi_s^x.\varphi_s$**
We can view every $S_\exists^s$ as a nondeterministic Büchi automaton with $\mathscr{L}(S_\exists^s) \subseteq ((\mathbb{N}^*)^m)^\omega$, i.e. the language contains infinite sequences of partial move vectors containing the moves for the universal players that are accepted. We now want to know for every state $s$ in the system $S_\exists$ what behavior of the universal players $p_{\pi_l}, 1 \leq l \leq m$ is allowed in the future if a run is in that state. So we consider the system $S_\exists^s$ for every state which is exactly the same system as $S_\exists$ but with state $s$ considered as the initial state.

Let $x$ be the number of states in $S_\exists$ and thus the number of existential quantifiers in every $\phi_s$. For every of these Büchi automata we calculate the HyperLTL formula $\varphi'_s = \exists\pi_s^1. \ldots \exists\pi_s^x.\varphi_s$ with the same language $\mathscr{L}(S_\exists^s) = \mathscr{L}(\varphi'_s)$. The formula $\varphi'_s$ can be constructed from the automaton $S_\exists^s$ by applying Construction 2.10.

$\mathscr{L}(\varphi'_s)$ describes which sequences of moves are allowed for the universal players starting from every state $s$ in order to preserve the possibility to win for the existential players. Note that $\mathscr{L}(S_\exists^s)$ is a language about move vector sequences. We can interpret this language as a language of traces by proceeding as follows:

Together with the states $s^{\pi_l}$ defined in $s = ((s^{\pi_1}, \ldots, s^{\pi_m}, s^{\pi'_1}, \ldots, s^{\pi'_n}), q)$ for $1 \leq l \leq m$ every sequence of move vectors in the language identifies $m$ paths trough the Kripke structure $K$, one for every universal player.

The language of $\varphi'_s$ can therefore be interpreted as a language of path tuples containing one path $u_{\pi_l}$ per universal player, where every path starts in the respective start state $s^{\pi_l}$. For all path tuples in this language the existential players can find corresponding paths such that the formula is fulfilled.

Because every path in $K$ defines a unique trace (applying the labeling function $l_K$ to every state in $u_{\pi_l}$ yields the corresponding trace) we can view $\varphi'_s$ as a formula describing a language of trace tuples, i.e. $\mathscr{L}(\varphi'_s) \subseteq ((2^{AP})^m)^\omega$.

Note that although we now loose information about the exact paths we do not loose any relevant information because HyperLTL formulas in general only consider traces represented in the trace assignments $\Pi$ and never reason about the actual underlying path.

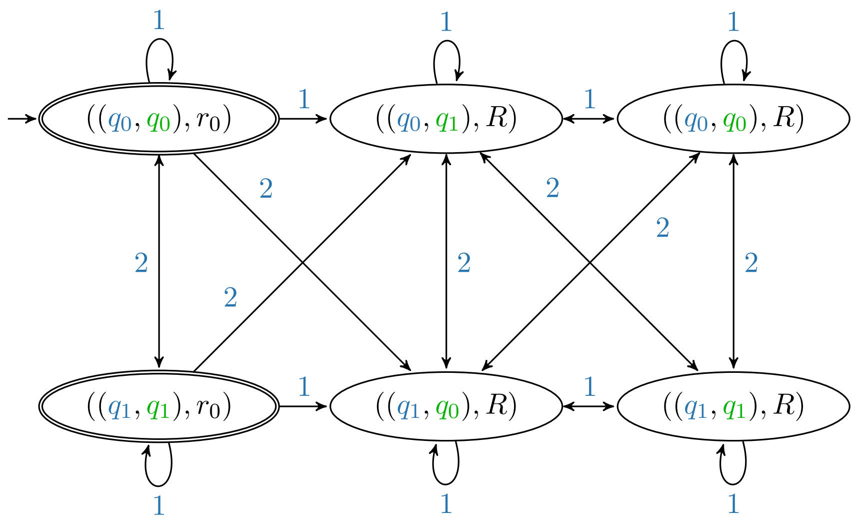


Figure 4.4: System $S_\exists$ built during Construction 4.4

The formulas $\varphi'_s$ cover exactly the information needed in order to compute adequate strategies for the existential players. We will thus use prophecy variables guessing the future value of these formulas in order to achieve completeness for safety formulas on finite state systems. $\square$

The following example illustrates the application of this construction.

**Example 4.5 (Application of Construction 4.4)**
Reconsider the problem whether the HyperLTL formula $\forall\pi_1.\exists\pi_2.\square(a_{\pi_1} \leftrightarrow a_{\pi_2})$ holds on the Kripke structure $K$ given in Figure 2.1.

**Step 1** requires to calculate the concurrent game structure $S$ from $K$ for the HyperLTL formula. This was done in Example 4.2 and the resulting concurrent game structure is given in Figure 4.2. Player $p_{\pi_1}$ is the universal player and $p_{\pi_2}$ is the existential player.

In **Step 2** the existential projection is applied to $S$ and we obtain the nondeterministic concurrent game structure $S_\exists$ as given in Figure 4.4 where the moves chosen by the existential player are projected away.

**Step 3:** For every state $s \in Q$ we consider the concurrent game structure $S^s_\exists$. This system then is interpreted as a nondeterministic Büchi automaton with $s$ being the initial state.

For every of these automata an equivalent HyperLTL formula $\varphi'_s$ is calculated using Construction 2.10.

These formulas $\varphi'_s$ describe the set of move sequences that is allowed for the universal player in order to ensure that the existential player has the possibility to win starting in state $s$. Every move sequence in the language of this HyperLTL formula identifies a unique path in the Kripke structure $K$ and every path generates a unique trace. Thus the HyperLTL formula $\varphi'_s$ gives the language of all traces that player $p_{\pi_1}$ is allowed to generate starting in the state $s$ that is currently examined. $\square$

We introduce prophecy variables $p_s$ to our formula and system as follows. For every state $s$ of the system $S_\exists$ we have computed the language $\varphi'_s$. A prophecy variable $p_s$ is now introduced for every $s$ to the set of atomic propositions and we require $p_s$ to be equivalent to $\varphi'_s$ at every point in time, i.e. $\square(p_s \leftrightarrow \varphi'_s)$. $p_s$ intuitively means that the universally quantified traces behave like it is required by the language of $\varphi'_s$. Note that in order to obtain a HyperLTL formula we have to pull the quantifiers to the front which is no problem here because the scope of the quantifiers does not change. Only the subformula $\varphi_s$ reasons about the traces quantified by $\exists\pi^1_s. \dots \exists\pi^x_s.$. The rest of the formula accesses different traces.

If a trace fulfills the requirements of such a language we know that there is a corresponding trace that allows us to satisfy the formula $\psi$. The $\varphi'_s$ are the HyperLTL formulas we were looking for in the application of our proof rule. There are only finitely many formulas $\varphi'_s$ namely one for every state $s \in (Q_K)^n \times Q_{A'}$ which is finite because $Q_K$ and $Q_{A'}$ are.

The prophecy variables are introduced as atomic propositions to the Kripke structure. For every possible truth value assignment of these variables one state is created. We will do this implicitly and only display the systems containing the original states. Nevertheless, we assume that for a given path through the system on the original states a path exists for every possible valuation of the prophecy variables considering the extended set of atomic propositions.

Formally the prophecy variables are introduced by Rule Prophecy presented in Figure 4.5. First we prove that this rule is sound and complete, i.e. the two formulas in the premise and conclusion of this rule are equivalent. Then we will have to show that in the resulting formula we have all information needed to give a winning set of strategies for the existential players on the self-composed system $S$ in order to prove that the proof system is complete.

**Theorem 4.6 (Soundness and Completeness of Rule Prophecy)**
The inference rule Prophecy from Figure 4.5 is sound and complete. $\square$

$$\frac{K \models \forall s \in Q_S.\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\exists\pi_s^1.\ldots\exists\pi_s^x.(\bigwedge_{s\in Q_S}\Box(p_s \leftrightarrow \varphi_s)) \to \psi}{K \models \forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\psi}$$

where $S$ is built as Construction 4.1 specifies, for all $s \in Q_S$ $p_s$ is fresh to AP and the formulas $\exists\pi_s^1.\ldots\exists\pi_s^x.\varphi_s$ are computed according to Construction 4.4

Figure 4.5: Rule Prophecy

**Proof** We prove the claim by showing that the two formulas of the rule are equivalent, i.e. we show the following:

$(\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\psi) \leftrightarrow$

$(\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\exists\pi_{\varphi_1}.\ldots\exists\pi_{\varphi_m}.(\bigwedge_{s\in Q_{S'}}\Box(p_s \leftrightarrow \varphi_s)) \to \psi)$.

"$\to$":

Assume $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\psi$ holds. We have to prove that then also the formula $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\exists\pi_{\varphi_1}.\ldots\exists\pi_{\varphi_m}.(\bigwedge_{s\in Q_{S'}}\Box(p_s \leftrightarrow \varphi_s)) \to \psi$ holds.

Following the equivalence $(\phi_1 \to \phi_2) \leftrightarrow (\neg\phi_1 \vee \phi_2)$ an implication holds whenever its premise is *false* or its conclusion is *true*. Note that $\psi$ in both formulas only reasons about the first $m + n$ quantified traces and the additional existential quantifiers in the second formula do not affect the truth value of $\psi$ at all. So because by the assumption we know that $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\psi$ holds we see that the conclusion of the implication in the second formula holds and thus followed directly from the assumption $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\exists\pi_{\varphi_1}.\ldots\exists\pi_{\varphi_x}.(\bigwedge_{s\in Q_{S'}}\Box(p_s \leftrightarrow \varphi_s)) \to \psi$ is *true*.

"$\leftarrow$":

Assume that $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\exists\pi_{\varphi_1}.\ldots\exists\pi_{\varphi_x}.(\bigwedge_{s\in Q_{S'}}\Box(p_s \leftrightarrow \varphi_s)) \to \psi$ holds. We want to show that $\forall\pi_1.\ldots\forall\pi_m.\exists\pi'_1.\ldots\exists\pi'_n.\psi$ holds too.

Let $\pi_l$ for $1 \le l \le m$ be arbitrary traces. Let $\pi'_k$ for $1 \le k \le n$ and $\pi_{\varphi_y}$ for $1 \le y \le x$ be the corresponding traces which have to exist by assumption.

- Whenever the premise $\bigwedge_{s\in Q_{S'}}\Box(p_s \leftrightarrow \varphi_s)$ holds we can immediately conclude that $\psi$ is *true* as well because we assume that the implication holds.

- In case $\bigwedge_{s \in Q_{S'}} \square(p_s \leftrightarrow \varphi_s)$ does not hold this means that there has to be at least one point in time where a prophecy variable $p_s$ guesses the wrong value for $\varphi_s$. Recall that the only constraint on the choice of the prophecy variables is that they have to be fresh to the set of atomic propositions $AP$. We can choose fresh prophecy variables $p'_s$ that always guess the right value of $\varphi_s$ and that then fulfill $\bigwedge_{s \in Q_{S'}} \square(p'_s \leftrightarrow \varphi_s)$. Again because the premise holds we know that the conclusion $\psi$ has to be *true*.

In both cases we can conclude that $\psi$ has to be *true* and by reintroducing the quantifiers we obtain $\forall \pi_1. \dots \forall \pi_m. \exists \pi'_1. \dots \exists \pi'_n. \psi$. So we can conclude that our claim holds and the Rule Prophecy is sound and complete. ∎

Note that in the proof above we did not make any assumptions about the kind of formula so that we can conclude that the rule Prophecy is sound and complete not only for safety formulas.

## 4.3 Computing Strategies

After the application of the Prophecy Rule 4.5 the resulting equivalent formula contains all needed information about the future explicitly. Then Rule 4.3 is applied to gain the concurrent game structure $S$ and using the information gained by the step before it is now possible to give the necessary strategies in order to prove that the resulting statement holds and the proof system is complete. We will now show how this set of strategies can be computed.

**Construction 4.7 (Calculate set of strategies)**
Let a Kripke structure $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ and a safety HyperLTL formula $\forall \pi_1. \dots \forall \pi_m. \exists \pi'_1. \dots \exists \pi'_n. \psi$ be given. Then we apply the Rule Prophecy 4.5 and afterwards the modified Rule Self-Composition 4.3. We obtain the concurrent game structure $S = (A, Q, AP, l, d, \delta)$ and the ATL* formula $\forall s \in Q. \langle\!\langle p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi^1_s}, \dots p_{\pi^x_s} \rangle\!\rangle ( \bigwedge_{s \in Q} \square(p_s \leftrightarrow \varphi_s)) \rightarrow \psi$.

Now we want to compute a set of strategies $F_{\{p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi^1_s}, \dots p_{\pi^x_s} | \forall s \in Q\}}$ containing one strategy for every player in the set annotated. All these strategies are memoryless because the whole information needed is encoded in the states due to the prophecy variables $p_s$.

We begin with computing the strategies $\sigma_{p_{\pi'_k}}$ for the existential players $p_{\pi'_k}$ with $1 \leq k \leq n$. The main idea is that we choose in every state $s$ a move $i$ for player $p_{\pi'_k}$ that preserves the possibility to win. Recall that we restricted our view to safety formulas which means that the set of strategies is winning if it guarantees that nothing bad will happen. If this is ensured from every state then there is no opportunity for any bad behavior such that the safety property is fulfilled and the strategies are winning.

To find an adequate move $i_k$ for a given player $p_{\pi'_k}$ in a given state $s = (s_t, q_t) = ((s_t^{\pi_1}, \dots, s_t^{\pi_m}, s_t^{\pi'_1}, \dots, s_t^{\pi'_n}), q_t)$ we compute the following for all possible values of $i_k$, i.e. for $1 \leq i_k \leq d_K(s_t^{\pi'_k})$:

$$L(s, i_k) = \forall\ 1 \leq l \leq m.\ \forall k < k^+ \leq n.$$
$$\bigvee_{1 < i_{k^+} \leq d_K(s_t^{\pi'_{k^+}})} \ \bigvee_{1 \leq j_l \leq d_K(s_t^{\pi_l})} \ \bigvee_{q_{t+1} \in Q_{t+1}} p_{s'}$$

where $s' = (s_{t+1}, q_{t+1}) = ((s_{t+1}^{\pi_1, j_1}, \dots, s_{t+1}^{\pi_m, j_m}, s_{t+1}^{\pi'_1, i_1}, \dots, s_{t+1}^{\pi'_n, i_n}), q_{t+1})$ and

- $\forall 1 \leq l \leq m.\ s_{t+1}^{\pi_l, j_l} = \delta_K(s_t^{\pi_l}, j_l)$
- $\forall 1 \leq k^- < k.\ s_{t+1}^{\pi'_{k^-}, i_{k^-}} = \delta_K(s_t^{\pi'_{k^-}}, \sigma_{p_{\pi'_{k^-}}}(s))$
- $s_{t+1}^{\pi'_k, i_k} = \delta_K(s_t^{\pi'_k}, i_k)$
- $\forall k < k^+ \leq n.\ s_{t+1}^{\pi'_{k^+}, i_{k^+}} = \delta_K(s_t^{\pi'_{k^+}}, i_{k^+})$
- $Q_{t+1} = \delta_{A'}(q_t, s_{t+1})$

Intuitively this formula checks whether there is a combination of moves chosen by the other players such that no bad behavior is caused by the resulting transition.

We choose any of the values $i_k$ for which $L(s, i_k)$ holds and define $\sigma_{p_{\pi'_k}}(s) = i_k$. This means that there was at least one *true* prophecy variable $p_{s'}$ checked by $L(s, i_k)$. This prophecy variable assures that taking the transition from $s$ to its successor state $s'$ does not cause any bad behavior and thus preserves the possibility to win. Moreover it guarantees that the universal players $p_{\pi_l}$ will choose adequate moves guaranteeing that their future behavior matches the language requirements of state $s'$ expressed as $\varphi_{s'}$. Adequate moves for the remaining existential players are then chosen using the same technique.

The strategies for the existential players $p_{\pi'_k}$ in a state $s$ have to be computed in an increasing order from $k = 1$ to $k = n$. When this is done for every $s \in Q$ we have found the desired strategies $\sigma_{p_{\pi'_k}}$.

Because we assume that the initial $\forall^*\exists^*$ HyperLTL formula holds on the given Kripke structure $K$ we also can conclude that in every step there is a corresponding move for the existential players that does not cause a violation of the safety requirements. If this was not the case at one point in time then the existential players had no possibility to win from that state and would not have chosen a move to go to that state one step before.

The strategies for the remaining players $p_{\pi_{\varphi_s^y}}$ for every $s \in Q$ and $1 \leq y \leq x$ may be chosen arbitrarily. In the case when the strategies for $p_{\pi_{\varphi_s^y}}$ are guessed correctly then the strategies of the players $p_{\pi'_k}$ guarantee that the formula $\psi$ is fulfilled. If the strategies were chosen wrong then this means that the premise does not hold anymore and the overall implication is fulfilled automatically. $\square$

## 4.4 Completeness Proof

Having this set of strategies we can show that the proof system for HyperLTL presented here is complete for finite state systems and safety formulas. This means if a safety HyperLTL formula $\varphi$ holds on a finite state Kripke structure $K$ and we apply Rule 4.5 and Rule 4.3 then the resulting ATL* formula $\psi$ holds on the generated concurrent game structure $S$. $S \models \psi$ is shown by proving that the strategies resulting from Construction 4.7 are winning. Together with the completeness of the ATL* proof system proven in [24] we then can conclude that our proof system is complete.

**Theorem 4.8 (Completeness of the Poof System)**
The presented proof system for HyperLTL is relative complete to the proof system of ATL* for finite state systems and safety formulas of the $\forall^*\exists^*$ fragment. $\square$

**Proof** To prove this claim we show that the statement $S \models true \rightarrow \forall s \in Q.\langle\langle p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi_s^1}, \dots p_{\pi_s^x}\rangle\rangle(\bigwedge_{s \in Q} \square(p_s \leftrightarrow \varphi_s)) \rightarrow \psi$ holds that means that there exists a set of strategies $F_\exists$ (generated by Construction 4.7) that is winning.

Let $K = (Q_K, q_0, d_K, \delta_K, AP_K, l_K)$ be a finite state Kripke structure and $\forall \pi_1. \dots \forall \pi_m. \exists \pi'_1. \dots \exists \pi'_n.\psi$ be a safety HyperLTL formula. Then we apply the Rule Prophecy 4.5 and afterwards the Modified Rule Self-Composition 4.3. Let $S'$ and $A'$ be the concurrent game structure and the nondeterministic Büchi automaton generated within this step. We obtain a finite state concurrent game structure $S = (A, Q, AP, l, d, \delta)$ and the ATL* formula $true \rightarrow \forall s \in Q.\langle\langle p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi_s^1}, \dots p_{\pi_s^x}\rangle\rangle(\bigwedge_{s \in Q} \square(p_s \leftrightarrow \varphi_s)) \rightarrow \psi$. Let $F_\exists = F_{\{p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi_s^1}, \dots p_{\pi_s^x} | \forall s \in Q\}}$ be the set of strategies resulting from running Construction 4.7.

Assume that $K \models \forall\pi_1.\dots\forall\pi_m.\exists\pi'_1.\dots\exists\pi'_n.\psi$.

We have to show that $F_\exists$ is a winning set of strategies. This means that for any choice of the strategies for the universal players $p_{\pi_l}$ with $1 \leq l \leq m$ the strategies in $F_\exists$ always guarantee that there is an accepting run through the nondeterministic concurrent game structure $S$. Thus the strategies together fix a unique path $u'$ through $S'$ and therefore define a set of paths $u$ through $S$ that only differ in their automaton component because the automaton $A'$ is nondeterministic on the input $u'$.

If $A'$ accepts $u'$ we know that there has to be an accepting run $r$ through $A'$. At the same time this means that $u'$ is in the language of $A'$ and thus by construction of $A'$ that the traces defined by the paths encoded in $u'$ satisfy the formula $\psi$, i.e. $\Pi_{u'} \models_K \psi$. Then we know that one of the paths $u$ through $S$, namely the one encoding the accepting run $r$ in the automaton component, is the accepting one that we were looking for and we can conclude that the strategies in $F_\exists$ are winning.

What is left to show is that the path $u'$ is in the language of $A'$ or equivalently that $\Pi_{u'} \models_K \psi$ assuming that the prophecy variables $p_s$ are chosen appropriately. Recall that $\psi$ is a safety formula and that therefore the existential players win if they can ensure that in every step nothing bad happens. If no step of a path causes any bad behavior then the safety property holds for this path. We show that the strategies in $F_\exists$ ensure exactly this.

By the assumption that $\forall\pi_1.\dots\forall\pi_m.\exists\pi'_1.\dots\exists\pi'_n.\psi$ holds on $K$ we know that starting in the initial state there has to be for every choice of the universal quantified players a move vector of the existential players that causes no bad behavior and thus preserves the possibility to win. The existential players then choose their moves in a given state $s$ according to Construction 4.7. Thus every player chooses a move for which $L(s,i)$ holds. That implies that there was at least one prophecy variable $p_{s'}$ that was $true$ where $s'$ is a successor state of $s$ in $S$. This prophecy variable intuitively guarantees that the universal players will behave in a way such that the existential players can choose adequate moves in order to satisfy the given safety property. As long as this possibility is given nothing bad has happened yet and thus a transition to the state $s'$ cannot cause any bad behavior. By doing this over and over again it is ensured that in every step good moves for the existential players are chosen meaning that no bad behavior occurs. The resulting path then satisfies the safety property.

Note that there always has to be at least one move $i$ for every existential player such that $L(s,i)$ is $true$. If this was not the case we would have no possibility to win in the current state $s$ and thus would have never gone there by construction.

Because the moves chosen by the strategies in $F_\exists$ ensure in every step that nothing bad will happen that violates the safety requirements expressed in the formula we can conclude that these strategies are winning. Thus we have shown that $S \models \forall s \in Q.\langle\langle p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi^1_s}, \dots p_{\pi^x_s}\rangle\rangle(\bigwedge_{s \in Q} \Box(p_s \leftrightarrow \varphi_s)) \rightarrow \psi$ as an adequate set of strategies $F_\exists$ exists. The following equivalence $\phi \leftrightarrow (true \rightarrow \phi)$ then implies that the statement we wanted to prove holds as well: $S \models true \rightarrow \forall s \in Q.\langle\langle p_{\pi'_1}, \dots p_{\pi'_n}, p_{\pi^1_s}, \dots p_{\pi^x_s}\rangle\rangle(\bigwedge_{s \in Q} \Box(p_s \leftrightarrow \varphi_s)) \rightarrow \psi$

Together with the completeness of the ATL* proof system that can be applied to this statement we have proven that the HyperLTL proof system presented is complete. ■

By adding the automaton $A'$ representing the safety LTL formula to the concurrent game structure and by the introduction of prophecy variables we were able to prove that the two sound Rules 4.5 and 4.3 together are complete for safety properties on finite state systems. This is the case because the prophecy variables cover all information that is needed in order to give an appropriate winning set of strategies for the resulting ATL* formula on the concurrent game structure. After that the ATL* proof system can be applied to the resulting formula. Because this proof system is sound and relative complete to the assertion language LTL we have obtained a sound and relative complete proof system for HyperLTL considering finite state systems and safety properties.

Chapter 5

# Conclusion

Within this thesis we developed a sound proof system for the $\forall^*\exists^*$ fragment of HyperLTL. When limited to safety HyperLTL formulas and finite state systems the proof system for HyperLTL presented is shown to be relative complete to the underlying proof system for ATL*.

In chapter 3 a first suggestion of a proof system was made. Using the concept of self-composition [5] several copies of the same system were combined in order to simultaneously represent different paths through that system (see Rule 3.3). To the resulting statement the ATL* proof system can be applied. We saw that this proof system is sound but not complete by giving a counterexample.

The relative completeness was achieved by introducing prophecy variables [1] that represent information about the future behavior of the universally quantified traces (see Rule 4.5). These variables contain explicit information about the future that is accessible in every state of the system. Using a product construction the automaton state is additionally introduced to the system representation (see Rule 4.3) in order to gain information about the accepted languages. Having these pieces of information it is possible to explicitly give a winning set of strategies for the quantified players in the resulting ATL* formula showing that the proof system is complete for safety formulas on finite state systems. The sound and relative complete proof system for ATL* is then applied to this ATL* formula and the generated concurrent game structure.

During the development of the proof system we illustrated on a small example how the different rules are applied. The problem we considered was how the valid statement $K \models \forall\pi_1.\exists\pi_2.\Box(a_{\pi_1} \leftrightarrow a_{\pi_2})$ can be proven using the presented proof system with $K$ being the Kripke structure given in Figure 2.1.

Having a proof system for a logic in general is beneficial because it allows the formal verification of every property that is expressible in that logic. A proof system for HyperLTL thus allows the verification of linear-time hyperproperties. Many interesting information-flow security policies like observational determinism are expressible in HyperLTL and can be proven on a given system model using this proof system. When using a proof system for a less expressive logic these properties cannot be examined in general. Nevertheless, some safety properties can be proven by applying the self-composition [5] as shown in [25] because the properties can be reduced to equivalent properties on the self-composed system. But overall, the set of properties that can be proven to hold on a system is extended by considering a HyperLTL proof system.

Using HyperLTL as a specification language for verification and synthesis tools allows to examine a wide range of desired system properties including interesting privacy and security constraints like information-flow properties. HyperLTL is similar to LTL which is nowadays used as a specification language in practice. Replacing this language by HyperLTL is therefore likely to be accepted by the actual users of these tools because they do not have to change completely but are already familiar and only have to integrate the concept of trace quantifiers in order to gain a lot of expressive power. This thesis is a first step towards this direction because it provides valuable insights into the formal verification of hyperproperties.

**Future Work**
The proof system as presented here is restricted to model checking finite state systems together with linear-time properties that are expressible in the $\forall^*\exists^*$ fragment of HyperLTL as a safety formula.

A next step would be to examine and prove the completeness of this proof system for liveness properties.

It would also be interesting to investigate the $\exists^*\forall^*$ fragment of HyperLTL as well as the fragments that contain more than one quantifier alternation. In the $\exists^*\forall^*$ case it is likely that the approach presented here only needs a minor adaption when the strategies for the existential players are computed. It might be sufficient to consider the conjunction of possible moves for the universal players instead of their disjunction. The adaption for the case with more than one quantifier alternation is possibly similar but there the additional question arises how the prophecy formulas $\varphi_s$ can be computed properly. Further research is needed to solve these challenges.

Another interesting problem that was not addressed in this thesis is the development of a proof system for HyperCTL*. As we have seen, HyperCTL* can express even more hyperproperties than HyperLTL can, because it uses path quantifiers and the quantifiers may occur at any position in the formula. By providing a proof system for this logic, all these hyperproperties could be formally proven on system models. This HyperCTL* proof system has to cope with the arbitrary position of the quantifiers. In our setting it might be possible to postpone the handling of this problem to the application of the ATL* proof system because the selective ATL* quantifiers may also occur at any position in the formula. In order to do that, there has to be a construction yielding an adequate ATL* formula for every HyperCTL* formula.

This bachelor's thesis provides a basis on which these open questions can be answered in future work.

# Bibliography

[1] Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theor. Comput. Sci.*, 82(2):253–284, May 1991.

[2] Bowen Alpern and Fred B. Schneider. Defining liveness. Technical report, Ithaca, NY, USA, 1984.

[3] Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. In *Journal of the ACM*, pages 100–109. IEEE Computer Society Press, 1997.

[4] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. *J. ACM*, 49(5):672–713, September 2002.

[5] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure Information Flow by Self-Composition. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, pages 100–114, 2004.

[6] Julius Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. *Internat. Congress on Logic, Methodology and Philosophy of Science*, 1960.

[7] Alonzo Church. Application of Recursive Arithmetic to the Problem of Circuit Synthesis.

[8] Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.

[9] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and Cesar Sanchez. Temporal Logics for Hyperproperties. In *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 265–284, 2014.

[10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[11] Stephen A. Cook. Soundness and Completeness of an Axiom System for Program Verification. *SIAM J. Comput.*, 7(1):70–90, 1978.

[12] E. Allen Emerson and Joseph Y. Halpern. Sometimes and Not Never revisited: on Branching versus Linear Time Temporal Logic. *J. ACM*, 33(1):151–178, 1986.

[13] Bernd Finkbeiner, Felix Klein, and Tobias Salzmann. Automata, Games and Verification. `https://www.react.uni-saarland.de/teaching/automata-games-verification-15/lecture-notes.html`, 2015. [Online; accessed 11-February-2016].

[14] Bernd Finkbeiner and Markus N. Rabe. The Linear-Hyper-Branching Spectrum of Temporal Logics. *it - Information Technology*, 56:273–279, November 2014.

[15] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL*. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 30–48, 2015.

[16] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[17] Zohar Manna and Amir Pnueli. Completing the Temporal Picture. *Theor. Comput. Sci.*, 83(1):91–130, 1991.

[18] Satoru Miyano and Takeshi Hayashi. Alternating Finite Automata on omega-Words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[19] Amir Pnueli. The Temporal Logic of Programs. *In Proc. of FOCS'77*, pages 46–57, 1977.

[20] Amir Pnueli and Yonit Kesten. A Deductive Proof System for CTL*. In *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, pages 24–40, 2002.

[21] R. M. Robinson. Restricted Set-Theoretical Definitions in Arithmetic. *Proc. Am. Math. Soc.*, pages 238–242, 1959.

[22] Aravinda Prasad Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems.* PhD thesis, Cambridge, MA, USA, 1983. AAI8403047.

[23] Matteo Slanina, Henny B. Sipma, and Zohar Manna. Proving ATL*

Properties of Infinite-State Systems. In *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, pages 242–256, 2006.

[24] Matteo Slanina, Henny B. Sipma, and Zohar Manna. Proving ATL* Properties of Infinite-State Systems. Technical Report REACT-TR-2006-02, Stanford University, Computer Science Department, REACT Group. 2006.

[25] Tachio Terauchi and Alexander Aiken. Secure Information Flow as a Safety Problem. In *Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings*, pages 352–367, 2005.

[26] Wolfgang Thomas. Handbook of Theoretical Computer Science (Vol. B). chapter Automata on Infinite Objects, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[27] Moshe Y. Vardi. Alternating Automata and Program Verification. In *Computer Science Today: Recent Trends and Developments*, pages 471–485. 1995.

[28] Mark Weiser. The Computer for the 21st Century. *Mobile Computing and Communications Review*, 3(3):3–11, 1999.