A Hierarchy for Accellera's Property Specification Language

Thomas Türk

May 1st, 2005

Diploma Thesis

University of Kaiserslautern

Supervisor:
Prof. Dr. Klaus Schneider

Vorliegende Diplomarbeit wurde von mir selbstständig verfasst. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.

Kaiserslautern, 1. Mai 2005

Thomas Türk

Contents

1 Introduction

1.1 Motivation

Model checking and equivalence checking are state-of-the-art in modern hardware circuit design. Moreover, standardised languages like the hardware description languages VHDL [4, 56] and Verilog [41] are widespread and allow one the convenient exchange of modules, which can also be sold as IP blocks. However, specifications of temporal properties that are required for model checking cannot be easily described with these languages [16, 45]. Hence, the research on model checking during the last two decades considered mainly temporal logics like LTL [42], CTL [21] and CTL* [22]. Moreover, several other formalisms like ω-automata [54] or monadic second order logics are used [32].

The syntax, the semantics and also the expressiveness and complexity of these temporal logics are very different. For example, while LTL model checking is PSPACE-complete, CTL model checking can be done in polynomial time, which is a consequence of the different expressive power of these logics. Moreover, temporal logics, ω-automata and monadic predicate logics form a hierarchy in terms of expressiveness [49].

The significant differences of the temporal logics used for specification prevent the use of different tools, a situation that was similar in digital circuit design before the standardisation of VHDL [4, 56] and Verilog [41]. For this reason, the increased importance of verification leads to industrial interest in standardised specification logics. Hence, great effort has been put on a standard of an industrial-strength property specification language. Accellera's Property Specification Language (PSL) [1, 2] is its result.

In this diploma thesis, a translation of a significant subset of PSL to classical temporal logic LTL is presented. It is well known how LTL can be translated to equivalent ω-automata [20, 24, 25, 52, 61]. In particular, there is a hierarchy of ω-automata [35, 36, 49, 58] that distinguishes between safety, liveness and four other classes of increasing expressiveness. Recently, the related subsets of LTL of this hierarchy have been syntactically characterised [48, 49]. In particular, linear-time translations have been presented [48, 49] that translate the temporal logic classes to corresponding ω-automata. The presented translation of a subset of PSL to LTL allows one to lift these temporal logic classes to PSL, i. e. it allows one to syntactically characterise subsets of PSL that can be translated to corresponding ω-automata.

Especially, subsets of PSL that can be translated to safety or liveness automata are interesting, since the classification into safety and liveness is important for many applications. Since only finite paths can be examined with simulation, this approach

is limited to safety and liveness properties. In particular, liveness properties can be proved and safety properties can be disproved by means of simulation. The same holds for most variants of bounded model checking [51], which are in some sense related to symbolic simulation. Other properties like fairness properties stating that a condition has to hold infinitely often, or persistence properties stating that a condition holds from a certain unknown point of time on, cannot be easily examined by simulation.

1.2 Main Objective

The main objective of this diploma thesis is to syntactically identify subsets of PSL that form a hierarchy similar to the well-known hierarchies of LTL and ω-automata. Furthermore, translations have to be developed that allow one to translate PSL classes to the corresponding LTL and ω-automata classes. Moreover, these translations and the hierarchy of PSL should be formally validated using an automated proof system.

The translation of the subset of PSL to ω-automata presented in this work consists of three parts: the translation of a subset of PSL to RLTL, the translation of RLTL to LTL and the translation of LTL to ω-automata (see Figure 1.1). RLTL is an extension of LTL by a reset operator that allows one to abort the consideration of a property. RLTL was introduced by Armoni, Bustan, Kupferman and Vardi in 2003 [5] to show the impact of different reset operators. As a consequence of [5], the semantics of the reset operator in PSL has been changed in version 1.1 to meet the semantics of the reset operator in RLTL. The previous version leads to a non-elementary blow-up in the translation to ω-automata. Thus, it is not surprising that a significant subset of PSL can be translated to RLTL. However, the actual translation of a subset of PSL to RLTL originates to this work. For the translation of RLTL to LTL, an algorithm presented in [5] is used. Finally, the translation of LTL to ω-automata as given by [48, 49] is used.

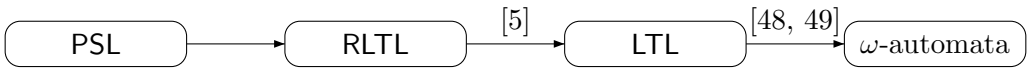

Figure 1.1: Translation of a subset of PSL to ω-automata

Using these translations between a subset of PSL, RLTL, LTL and ω-automata, the hierarchy of LTL identified by Klaus Schneider [48, 49] is lifted to PSL and RLTL, i. e. PSL and RLTL classes are identified that are translated to the corresponding LTL class by the presented translations. Therefore, they can also be translated to the corresponding class of ω-automata [49]. Moreover, the identified PSL and RLTL classes are complete in the sense, that they are as expressive as the corresponding LTL class. Especially, PSL classes are identified that correspond to liveness and safety properties.

In 2004, Claessen and Mårtensson presented a subset of PSL, called *weak* PSL that can express only liveness properties [18]. The PSL class corresponding to liveness properties defined in this work is different from weak PSL. In contrast to weak PSL, negations are considered. On the other hand, in contrast to the classes defined in this

work, weak PSL may contain certain regular expressions.

The example of the slightly different definitions of the reset operator in PSL version 1.01 and version 1.1 and the enormous impact of this small change shows that proving the correctness of the translations is quite tricky. PSL is a complex language and that includes many special cases. Therefore, all parts of the translation of PSL to ω-automata are formally validated in the interactive proof system for higher order logic HOL [27, 29]. Moreover, the identified hierarchy of PSL is formally validated, too. The resulting HOL library can be found on the included CD.

There is already a deep embedding of PSL in HOL [26]. This embedding is not part of the Accellera standard. However, some members of the Accellera Formal Property Language Technical Committee reviewed this embedding. In addition to this PSL library, some parts of an existing LTL library [50] are used. There are of course also other embeddings of temporal logics in HOL like [34, 57], but these are not useful in this context, since they provide no deep embeddings.

In the scope of this diploma thesis, RLTL, LTL and ω-automata have been deeply embedded in HOL. Using Mike Gordon's deep embedding of PSL and these embeddings of RLTL, LTL and ω-automata, the correctness of the entire translation of PSL to ω-automata is formally proved. Moreover, it is proved in HOL that the presented translation from PSL to LTL translates a PSL class to the corresponding LTL class. Since this LTL class can be translated to the corresponding class of ω-automata, the translation of PSL to LTL allows one to translate the PSL class to the corresponding ω-automata class.

The usage of Mike Gordon's PSL library is an additional value of this work, because it is one of the first applications of this library. Translating PSL to another language should find other bugs in the PSL library than sanity checking inside the PSL library itself. Indeed, during this diploma thesis a small, until then unknown bug has been discovered in the embedding, which will be mentioned later.

1.3 Document Structure

This diploma thesis is organised as follows: In Chapter 2, the used formalisms LTL, RLTL, PSL and ω-automata are introduced. Based on these introductions, the translations between these formalisms are presented in Chapter 3. Since there are formal HOL-proofs of the presented translations, only the main ideas of the proofs are given in the thesis, while machine-checked proofs are given on the included CD. In Chapter 4, the translations are used to define PSL classes that correspond to the temporal logic hierarchy [36, 48, 49] and hence, to the ω-automaton hierarchy [35, 49, 58]. Chapter 5 describes the work done in HOL. The embeddings of the used formalisms, the overall structure of the theories and important lemmata are described. However, formal proofs are not presented here either. Instead of this, this section is meant to give an overview about HOL and the HOL theories created during this work. Finally, some conclusions are drawn and directions for future work are discussed.

2 Basics

In this chapter, the formalisms LTL, RLTL, PSL and ω-automata that are essential for this work are introduced. All these formalisms are used to describe the temporal behaviour of a system. Therefore, they share some common elements. They all use propositional logic to describe properties of the current point of time, and they all describe properties of a sequence of points of time, a so-called *path*. Thus, the used notations for propositional logic and paths are defined first:

Definition 2.0.1 (Propositional Logic)
Let $\mathcal{V}$ be a set of variables. Then, the set of propositional formulas over $\mathcal{V}$ *(short $\mathsf{prop}_\mathcal{V}$) is recursively given as follows:*

- *each variable $v \in \mathcal{V}$ is a propositional formula*
- $\neg\varphi \in \mathsf{prop}_\mathcal{V}$, *if* $\varphi \in \mathsf{prop}_\mathcal{V}$
- $\varphi \wedge \psi \in \mathsf{prop}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{prop}_\mathcal{V}$

An assignment *over $\mathcal{V}$ is a subset of $\mathcal{V}$. In our context, assignments are also called* states. *The set of all states over $\mathcal{V}$, which is the power set of $\mathcal{V}$, is denoted by $\mathcal{P}(\mathcal{V})$. The semantics of a propositional formula with respect to a state s is given by the relation $\models_\mathsf{prop}$ that is defined as follows:*

- $s \models_\mathsf{prop} v$ *iff* $v \in s$
- $s \models_\mathsf{prop} \neg\varphi$ *iff* $s \not\models_\mathsf{prop} \varphi$
- $s \models_\mathsf{prop} \varphi \wedge \psi$ *iff* $s \models_\mathsf{prop} \varphi$ *and* $s \models_\mathsf{prop} \psi$

If $s \models_\mathsf{prop} \varphi$ holds, then the assignment s is said to fulfil *(or to* model*) the propositional formula φ.*

For reasons of simplicity, the operator $\wedge$ is often omitted. For example, x_1x_2 means $x_1 \wedge x_2$. Additionally, further propositional operators like $\vee, \rightarrow, \leftrightarrow$ etc. are added as *syntactic sugar*, i. e. they are added as shorthands for formulas not containing these operators:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$

- $\varphi \leftrightarrow \psi := \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- $\mathsf{true} := v \vee \neg v$ for an arbitrary variable $v \in \mathcal{V}$
- $\mathsf{false} := \neg\mathsf{true}$

Definition 2.0.2 (Words)
A finite word v over a set Σ of length $|v| = n + 1$ is a function $v : \{0, \dots n\} \rightarrow \Sigma$. An infinite word v over Σ is a function $v : \mathbb{N} \rightarrow \Sigma$. Its length is denoted by $|v| = \infty$. The set Σ is called alphabet. *The elements of Σ are called* letters. *The finite word of length 0 is called the* empty word *(denoted by ε). For reasons of simplicity, $v(i)$ is often denoted by v^i for $i \in \mathbb{N}$. Using this notation, words are often given in the form $v^0 v^1 v^2 \dots v^n$ or $v^0 v^1 \dots$. The set of all finite words over Σ is denoted by Σ^*, and the set of all infinite words over Σ is denoted by Σ^ω.*

Counting of letters starts with zero, i. e. v^{i-1} refers to the i-th letter of v. Furthermore, $v^{i\cdot\cdot}$ denotes the suffix of v starting at position i, i. e. $v^{i\cdot\cdot} = v^i v^{i+1} \dots$ for all $i < |v|$. The finite word $v^i v^{i+1} \dots v^j$ is denoted by $v^{i\cdot\cdot j}$. Notice that in case $j < i$ the expression $v^{i\cdot\cdot j}$ evaluates to the empty word ε. For two words v_1, v_2, we use $v_1 v_2$ for the concatenation of v_1 and v_2. Finally, we use l^ω for the infinite word v with $v^j = l$ for all j.

Definition 2.0.3 (Paths)
Temporal logics are often used to reason about the behaviour of transition systems. A path *of a transition systems is a word, for which is possible to transition from one letter of the word to the next letter. However, the terminology of* PSL *does not distinguish between paths and words [2]. Therefore, the terms 'path' and 'word' are used synonymously in this work.*

2.1 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) has been introduced by Pnueli in 1977 [42]. Essentially, it consists of propositional logic enriched with the temporal operators X and U. The formula $\mathsf{X}\varphi$ means that the property φ holds at the next point of time, $\varphi \,\mathsf{U}\, \psi$ means that φ holds until ψ holds and that ψ eventually holds. The operators $\overleftarrow{\mathsf{X}}$ and $\overleftarrow{\mathsf{U}}$ express the same properties for the past instead of the future. Therefore, the operators X and U are called *future operators*, while $\overleftarrow{\mathsf{X}}$ and $\overleftarrow{\mathsf{U}}$ are called *past operators*. LTL without past operators is as expressive as LTL with past operators [23]. Hence, LTL is often regarded as the presented logic without past operators. In this work, this subset is called FutureLTL. This work mainly considers FutureLTL, past operators are not needed. However, they are introduced and the presented translation of LTL to ω-automata considers these operators, because LTL with past operators is exponentially more succinct than LTL without past operators [38].

Definition 2.1.1 (Syntax of Linear Temporal Logic (LTL))
The following mutually recursive definitions introduce the set $\mathsf{ltl}_{\mathcal{V}}$ *of* LTL *formulas over a given set of variables* $\mathcal{V}$*:*

- *each propositional formula* $p \in \mathsf{prop}_{\mathcal{V}}$ *is a* LTL *formula*
- $\neg\varphi$, $\varphi \wedge \psi \in \mathsf{ltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{ltl}_{\mathcal{V}}$
- $\mathsf{X}\varphi$, $\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{ltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{ltl}_{\mathcal{V}}$
- $\overleftarrow{\mathsf{X}}\varphi$, $\varphi \,\overleftarrow{\underline{\mathsf{U}}}\, \psi \in \mathsf{ltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{ltl}_{\mathcal{V}}$

Definition 2.1.2 (Future Fragment of LTL (FutureLTL))
The following definitions formally introduce the future fragment of LTL*. For a given set of variables* $\mathcal{V}$*, the set* $\mathsf{fltl}_{\mathcal{V}}$ *is the set of* FutureLTL *formulas over* $\mathcal{V}$*. It is given by the following mutually recursive definitions:*

- *each propositional formula* $p \in \mathsf{prop}_{\mathcal{V}}$ *is a* FutureLTL *formula*
- $\neg\varphi$, $\varphi \wedge \psi \in \mathsf{fltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{fltl}_{\mathcal{V}}$
- $\mathsf{X}\varphi$, $\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{ltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{fltl}_{\mathcal{V}}$

Notice, that $\mathsf{fltl}_{\mathcal{V}}$ *is a proper subset of* $\mathsf{ltl}_{\mathcal{V}}$*.*

Definition 2.1.3 (Semantics of Linear Temporal Logic (LTL))
For $b \in \mathsf{prop}_{\mathcal{V}}$ *and* $\varphi, \psi \in \mathsf{ltl}_{\mathcal{V}}$ *the semantics of* LTL *with respect to an infinite word* $v \in \mathcal{P}(\mathcal{V})^\omega$ *and a point of time* $t \in \mathbb{N}$ *is given by:*

- $v \models^t_{\mathsf{ltl}} b$ *iff* $v^t \models_{\mathsf{prop}} b$
- $v \models^t_{\mathsf{ltl}} \neg\varphi$ *iff* $v \not\models^t_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi \wedge \psi$ *iff* $v \models^t_{\mathsf{ltl}} \varphi$ *and* $v \models^t_{\mathsf{ltl}} \psi$
- $v \models^t_{\mathsf{ltl}} \mathsf{X}\varphi$ *iff* $v \models^{t+1}_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi \,\underline{\mathsf{U}}\, \psi$ *iff* $\exists k.\ k \geq t \ \wedge\ v \models^k_{\mathsf{ltl}} \psi \ \wedge\ \forall j.\ t \leq j < k \rightarrow v \models^j_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \overleftarrow{\mathsf{X}}\varphi$ *iff* $t > 0 \ \wedge\ v \models^{t-1}_{\mathsf{ltl}} \varphi$
- $v \models^t_{\mathsf{ltl}} \varphi \,\overleftarrow{\underline{\mathsf{U}}}\, \psi$ *iff* $\exists k.\ k \leq t \ \wedge\ v \models^k_{\mathsf{ltl}} \psi \ \wedge\ \forall j.\ k < j \leq t \rightarrow v \models^j_{\mathsf{ltl}} \varphi$

Furthermore, a word $v \in \mathcal{P}(\mathcal{V})^\omega$ *is said to* satisfy *(or to* model*) a* LTL *formula* $\varphi \in \mathsf{ltl}_{\mathcal{V}}$ *(short* $v \models_{\mathsf{ltl}} \varphi$*) iff* $v \models^0_{\mathsf{ltl}} \varphi$*. A* LTL *formula* φ *is* equivalent *to a* LTL *formula* ψ *(denoted by* $\varphi \equiv_{\mathsf{ltl}} \psi$*) if for all words* $v \in \mathcal{P}(\mathcal{V})^\omega$ *and all points of time* $t \in \mathbb{N}$ *the relation* $v \models^t_{\mathsf{ltl}} \varphi$ *holds iff* $v \models^t_{\mathsf{ltl}} \psi$ *holds. If for all* v *the relation* $v \models^0_{\mathsf{ltl}} \varphi$ *holds iff* $v \models^0_{\mathsf{ltl}} \psi$ *holds,* φ *and* ψ *are* initially equivalent *(denoted by* $\varphi \equiv^0_{\mathsf{ltl}} \psi$*).*

LTL is usually enriched with syntactic sugar. In this work, the operators $\vee, \rightarrow$ and $\leftrightarrow$ are used with their usual definition. Additionally, the operators F, G, U and B are important:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- $\mathsf{F}\varphi := \mathsf{true}\ \underline{\mathsf{U}}\ \varphi$
- $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$
- $\varphi\ \mathsf{U}\ \psi := \varphi\ \underline{\mathsf{U}}\ \psi\ \vee\ \mathsf{G}\varphi$
- $\varphi\ \mathsf{B}\ \psi := \neg(\neg\varphi\ \underline{\mathsf{U}}\ \psi)$

$\mathsf{F}\varphi$ means that there is a point of time in the future where φ holds. $\mathsf{G}\varphi$ means that φ holds from now on. $\varphi\ \mathsf{U}\ \psi$ is a weak variant of $\varphi\ \underline{\mathsf{U}}\ \psi$: In contrast to $\varphi\ \underline{\mathsf{U}}\ \psi$, it does not demand that φ eventually holds; in this case φ must always hold. $\varphi\ \mathsf{B}\ \psi$ means that either ψ never holds or that there is a point of time before ψ holds, where φ holds.

The operators F and G will become important for defining classes of ω-automata. The U operator is used to explain the translation of LTL to ω-automata. Finally, B is introduced, because it is the last needed operator to be able to transform FutureLTL formulas into negation normal form [49]. However, this normal form is not discussed here.

2.2 Reset Linear Temporal Logic (RLTL)

RLTL is an extension of FutureLTL by a reset operator, which allows one to abort the consideration of a property. As mentioned above, RLTL has been introduced to show the impact of different reset operators [5].

Definition 2.2.1 (Syntax of Reset Linear Temporal Logic (RLTL))
The following mutually recursive definitions introduce the set $\mathsf{rltl}_{\mathcal{V}}$ *of* RLTL *formulas over a given set of variables* $\mathcal{V}$*:*

- *each propositional formula* $p \in \mathsf{prop}_{\mathcal{V}}$ *is a* RLTL *formula*
- $\neg\varphi,\ \varphi \wedge \psi \in \mathsf{rltl}_{\mathcal{V}}$, *if* $\varphi, \psi \in \mathsf{rltl}_{\mathcal{V}}$
- $\mathsf{X}\varphi,\ \varphi\ \underline{\mathsf{U}}\ \psi \in \mathsf{rltl}$, *if* $\varphi, \psi \in \mathsf{rltl}_{\mathcal{V}}$
- $\mathsf{ACCEPT}(\varphi, b) \in \mathsf{rltl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{rltl}_{\mathcal{V}}$, $b \in \mathsf{prop}_{\mathcal{V}}$

Some operators like $\neg, \wedge$ or $\underline{\mathsf{U}}$ are used by several formalisms discussed in this work. In most cases, it is clear by the context or does not make any difference to which formalism one of these operators belongs. If it is important, it is denoted by a subscript, e. g. $\neg_{\mathsf{prop}}, \neg_{\mathsf{ltl}}, \neg_{\mathsf{rltl}}$ or $\wedge_{\mathsf{prop}}$ are used. In the case of RLTL formulas, there are special cases, where it matters, whether the propositional negation $\neg_{\mathsf{prop}}$ or the negation of RLTL $\neg_{\mathsf{rltl}}$ is used. An example is presented below.

Definition 2.2.2 (Semantics of Reset Linear Temporal Logic (RLTL))
The semantics of LTL *is given with respect to a word v and a point of time t. To define the semantics of* RLTL*, an* acceptance condition $a \in \mathsf{prop}_{\mathcal{V}}$ *and a* rejection condition $r \in \mathsf{prop}_{\mathcal{V}}$ *are additionally needed. These conditions are used to capture the required information about* ACCEPT *operators in the context of the formula. Thus, for $b \in \mathsf{prop}_{\mathcal{V}}$ and $\varphi, \psi \in \mathsf{rltl}_{\mathcal{V}}$, the semantics of* RLTL *with respect to an infinite word $v \in \mathcal{P}(\mathcal{V})^\omega$, acceptance / rejection conditions $a, r \in \mathsf{prop}_{\mathcal{V}}$ and a point of time $t \in \mathbb{N}$ is given by:*

- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} b$ *iff* $v^t \models_{\mathsf{prop}} a$ *or* $\left(v^t \models_{\mathsf{prop}} b \text{ and } v^t \not\models_{\mathsf{prop}} r\right)$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \neg\varphi$ *iff* $\langle v, r, a\rangle \not\models^t_{\mathsf{rltl}} \varphi$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \wedge \psi$ *iff* $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi$ *and* $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \psi$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \mathsf{X}\varphi$ *iff* $v^t \models_{\mathsf{prop}} a$ *or* $\left(\langle v, a, r\rangle \models^{t+1}_{\mathsf{rltl}} \varphi \text{ and } v^t \not\models_{\mathsf{prop}} r\right)$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \,\underline{\mathsf{U}}\, \psi$ *iff* $\exists k.\ k \geq t \ \wedge\ \langle v, a, r\rangle \models^k_{\mathsf{rltl}} \psi \ \wedge$
  $\forall j.\ t \leq j < k \rightarrow \langle v, a, r\rangle \models^j_{\mathsf{rltl}} \varphi$
- $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \mathsf{ACCEPT}(\varphi, b)$ *iff* $\langle v, a \vee (b \wedge \neg r), r\rangle \models^t_{\mathsf{rltl}} \varphi$

A word $v \in \mathcal{P}(\mathcal{V})^\omega$ is said to satisfy a RLTL *formula $\varphi \in \mathsf{rltl}_{\mathcal{V}}$ (short $v \models_{\mathsf{rltl}} \varphi$) iff $\langle v, \mathsf{false}, \mathsf{false}\rangle \models^0_{\mathsf{rltl}} \varphi$ holds. A* RLTL *formula φ is* equivalent *to a* RLTL *formula ψ (denoted by $\varphi \equiv_{\mathsf{rltl}} \psi$) if for all words $v \in \mathcal{P}(\mathcal{V})^\omega$ the relation $v \models_{\mathsf{rltl}} \varphi$ holds iff $v \models_{\mathsf{rltl}} \psi$ holds.*

$\mathsf{ACCEPT}(\varphi, b)$ stops the evaluation of the formula φ, when the condition b holds and accepts formulas that have not been rejected before. For example, the word $\{a\}\{c\}\emptyset^\omega$ does not satisfy the RLTL formula $a \,\underline{\mathsf{U}}\, b$, but it satisfies $\mathsf{ACCEPT}(a \,\underline{\mathsf{U}}\, b, c)$. On the other hand, the word $\emptyset\{c\}\emptyset^\omega$ does not satisfy $\mathsf{ACCEPT}(a \,\underline{\mathsf{U}}\, b, c)$, since $a \,\underline{\mathsf{U}}\, b$ fails before c occurs. To understand the impact of the acceptance and rejection conditions and thus, to understand the semantics of the ACCEPT operator, the following lemma is important:

Lemma 2.2.3 *For all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$, all formulas $\varphi \in \mathsf{rltl}_{\mathcal{V}}$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_{\mathcal{V}}$ and all points of time $t \in \mathbb{N}$, the following holds:*

$$\begin{array}{lcll} \left(v^t \models_{\mathsf{prop}} a \ \wedge\ v^t \not\models_{\mathsf{prop}} r\right) & \Longrightarrow & \langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi & \text{and} \\ \left(v^t \not\models_{\mathsf{prop}} a \ \wedge\ v^t \models_{\mathsf{prop}} r\right) & \Longrightarrow & \langle v, a, r\rangle \not\models^t_{\mathsf{rltl}} \varphi & \end{array}$$

This lemma can be easily proved by structural induction. A proof is omitted here, but can be found in the HOL theories[1] on the included CD. Lemma 2.2.3 states that if the acceptance condition holds, every formula is accepted. On the other hand, if the rejection condition holds, every formula is rejected. Neither the acceptance nor the rejection condition is dominant, but: If for a word $v \in \mathcal{P}(\mathcal{V})^\omega$, the acceptance condition $a \in \mathsf{prop}_\mathcal{V}$ and the rejection condition $r \in \mathsf{prop}_\mathcal{V}$ hold both at time $t \in \mathbb{N}$, i. e. in case $v^t \models_{\mathsf{prop}} a$ and $v^t \models_{\mathsf{prop}} r$, the truth of the expression $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi$ depends on the formula $\varphi \in \mathsf{rltl}_\mathcal{V}$. With $a, b, c \in \mathcal{V}$ for example, $\langle \{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} c$ holds, but $\langle \{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} \neg_{\mathsf{rltl}} c$ does not hold. Furthermore, this example shows that it may be important whether the propositional negation $\neg_{\mathsf{prop}}$ or the RLTL negation $\neg_{\mathsf{rltl}}$ is used: $\langle \{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} \neg_{\mathsf{prop}} c$ holds. The example may lead to the conjecture that in case $v^t \models_{\mathsf{prop}} a$ and $v^t \models_{\mathsf{prop}} r$, the relation $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi$ holds iff $\langle v, a, r\rangle \not\models^t_{\mathsf{rltl}} \neg_{\mathsf{rltl}} \varphi$ holds. However, this conjecture is wrong, as the following example shows[2]: $\langle \{a,b\}\{a\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} c \,\underline{\mathsf{U}}\, (\neg_{\mathsf{rltl}} c)$ and $\langle \{a,b\}\{a\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} \neg_{\mathsf{rltl}} \big(c \,\underline{\mathsf{U}}\, (\neg_{\mathsf{rltl}} c)\big)$ hold.

In general, the case that the acceptance and the rejection condition hold at the same point of time causes a lot of problems. Luckily, all pairs of acceptance / rejection conditions (a, r) considered for evaluating expressions of the form $v \models_{\mathsf{rltl}} \varphi$ have the property $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$, because the initial pair $(\mathsf{false}, \mathsf{false})$ possesses this property, and because the pair of acceptance / rejection conditions is only changed by the occurrence of a $\neg_{\mathsf{rltl}}$ or ACCEPT operator in φ. The semantics of these operators preserve the property $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$.

Therefore, $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$ or weaker variants of this condition are often assumed when talking about pairs of acceptance / rejection conditions (a, r). These assumptions simplify some proofs, because special cases (as the ones mentioned above) can be excluded. For example, it does not matter if $\neg_{\mathsf{prop}}$ or $\neg_{\mathsf{rltl}}$ is used, if $\forall s.\, s \models_{\mathsf{prop}} \neg(a \wedge r)$ is assumed[3]. A assumption of this kind is also used by the following lemma:

Lemma 2.2.4 *For all infinite words $v_1, v_2 \in \mathcal{P}(\mathcal{V})^\omega$, all formulas $\varphi \in \mathsf{rltl}_\mathcal{V}$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following holds[4]:*

$$\begin{aligned}
&\Big(\exists k.\ k \geq t \ \wedge\ v_1^{t..k-1} = v_2^{t..k-1} \ \wedge \\
&\quad \Big(\big(v_1^k \models_{\mathsf{prop}} a \ \wedge\ v_2^k \models_{\mathsf{prop}} a \ \wedge\ v_1^t \not\models_{\mathsf{prop}} r \ \wedge\ v_2^t \not\models_{\mathsf{prop}} r\big) \vee \\
&\qquad \big(v_1^k \not\models_{\mathsf{prop}} a \ \wedge\ v_2^k \not\models_{\mathsf{prop}} a \ \wedge\ v_1^t \models_{\mathsf{prop}} r \ \wedge\ v_2^t \models_{\mathsf{prop}} r\big)\Big)\Big) \Longrightarrow \\
&\Big(\langle v_1, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \Longleftrightarrow \langle v_2, a, r\rangle \models^t_{\mathsf{rltl}} \varphi\Big)
\end{aligned}$$

Lemma 2.2.4 states that the semantics of RLTL only considers the part of a word from the current point of time to an occurrence of either the acceptance or the rejection

[1] theorem `RLTL_ACCEPT_REJECT_THM` in theory `ResetLTL_Lemmata`
[2] theorem `RLTL_SEM_TIME___STRANGE_NEGATION_EXAMPLE` in theory `ResetLTL_Lemmata`
[3] theorem `RLTL_SEM_PROP_RLTL_OPERATOR_EQUIV` in theory `ResetLTL`
[4] theorem `RLTL_EQUIV_PATH_STRONG_THM` in theory `ResetLTL_Lemmata`

condition. Again, the occurrence of both conditions at the same point of time, may cause trouble. It is important that the 'or' is exclusive as the following example shows: $\langle\{a,b\}\emptyset^\omega, a, b\rangle \models^0_{\mathsf{rltl}} a \underline{\mathsf{U}} \neg_{\mathsf{rltl}} c$, but $\langle\{a,b\}\{c\}\emptyset^\omega, a, b\rangle \not\models^0_{\mathsf{rltl}} a \underline{\mathsf{U}} \neg_{\mathsf{rltl}} c$.

The other RLTL operators have the same semantics as the corresponding LTL operators. As well, most of the used syntactic sugar is similar to the syntactic sugar of LTL:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- $\mathsf{F}\varphi := \mathsf{true}\ \underline{\mathsf{U}}\ \varphi$
- $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$
- $\varphi\ \mathsf{U}\ \psi := \varphi\ \underline{\mathsf{U}}\ \psi\ \vee\ \mathsf{G}\varphi$
- $\varphi\ \mathsf{B}\ \psi := \neg(\neg\varphi\ \underline{\mathsf{U}}\ \psi)$
- $\mathsf{REJECT}(\varphi, b) = \neg(\mathsf{ACCEPT}(\neg\varphi, b))$

In addition to the syntactic sugar of LTL, a dual operator for the ACCEPT operator is defined. Using this REJECT operator and the other operators defined as syntactic sugar, it is possible to transform every RLTL formula to a formula, that contains no $\neg_{\mathsf{rltl}}$ operators.

Lemma 2.2.5 *For every* RLTL *formula* $\varphi \in \mathsf{rltl}_\mathcal{V}$*, there is an equivalent* RLTL *formula* φ_{NNF} *in* negation normal form*, i. e.* φ_{NNF} *contains no* $\neg_{\mathsf{rltl}}$ *operators. For an arbitrary* RLTL *formula* φ*, the formula* $\mathcal{N}(\varphi)$ *is such an equivalent formula in negation normal form*[5]*. Thereby, the rewrite relation* $\mathcal{N}$ *is defined as follows:*

- $\mathcal{N}(\neg_{\mathsf{rltl}} b) := \neg_{\mathsf{prop}} b$
- $\mathcal{N}\big(\neg(\neg\varphi)\big) := \mathcal{N}(\varphi)$
- $\mathcal{N}\big(\neg(\varphi \wedge \psi)\big) := \mathcal{N}(\neg\varphi) \vee \mathcal{N}(\neg\psi)$
- $\mathcal{N}\big(\neg(\mathsf{X}\varphi)\big) := \mathsf{X}\big(\mathcal{N}(\neg\varphi)\big)$
- $\mathcal{N}\big(\neg(\varphi\ \underline{\mathsf{U}}\ \psi)\big) := \mathcal{N}(\neg\varphi)\ \mathsf{B}\ \mathcal{N}(\psi)$
- $\mathcal{N}\big(\neg(\mathsf{ACCEPT}(\varphi, b))\big) := \mathsf{REJECT}(\mathcal{N}(\neg\varphi), b)$

[5] theorem `RLTL_NEGATION_NORMAL_FORM` in theory `ResetLTL_Lemmata`

All $\varphi \in \mathsf{rltl}_{\mathcal{V}}$ are equivalent to $\mathcal{N}(\varphi)$, i. e. for all words $v \in \mathcal{P}(\mathcal{V})^\omega$ the relation $v \models_{\mathsf{rltl}} \varphi$ holds iff $v \models_{\mathsf{rltl}} \mathcal{N}(\varphi)$ holds. However, $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi$ iff $\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \mathcal{N}(\varphi)$ does not hold in general. It has already been discussed that it may be important if $\neg_{\mathsf{rltl}}$ or $\neg_{\mathsf{prop}}$ is used. Thus, subformulas of the form $\neg_{\mathsf{rltl}} b$ are problematic. However, also subformulas of the form $\neg(\mathsf{X}\varphi)$ cause problems. For formulas of these forms, it is required that the acceptance and rejection condition do not hold at the same time to ensure that the semantic of a formula is not changed by $\mathcal{N}$.

2.3 Accellera's Property Specification Language (PSL)

As mentioned above, PSL is a standardised industrial-strength property specification language. PSL was chartered by the Functional Verification Technical Committee of Accellera. The Sugar language from IBM was chosen as the basis for PSL [1, 6]. The Language Reference Manual for PSL version 1.0 [1] was released in April 2003. Finally, in June 2004 version 1.1 [2] was released, where some anomalies were corrected.

PSL is designed as an input language for formal verification and simulation tools as well as a language for documentation. Therefore, it has to be easy to read, and at the same time, it must be precise and highly expressive. In particular, PSL contains features for simulation like finite paths, features for hardware specification like clocked statements and a lot of syntactic sugar.

PSL consists of four layers: The Boolean one, the temporal one, the verification one and the modelling one. The Boolean layer is used to construct expressions that can be evaluated in one state. The temporal layer is the heart of the language. It is used to express properties concerning more than one state, i. e. temporal properties. The temporal layer is divided into the *Foundation Language* (FL) and the *Optional Branching Extension* (OBE). FL is, like LTL, a linear time temporal logic. Each point of time is assumed to have only one successor. Therefore, the input is represented as a word of points of time. In contrast, OBE is a branching time temporal logic. A point of time may have more than one possible successor. Therefore, the input is a tree of points of time, i. e. there may be more than one possible future. With OBE, it is possible for example to express properties like For all points of time of each possible future, there is a possible future such that there is a point of time in this future, when some property holds. The verification layer is used to tell tools, what to do with the properties expressed by the temporal layer. Finally, the modelling layer is used to describe assumptions about the behaviour of inputs and to model auxiliary hardware that is not part of the design.

Additionally, PSL comes in four flavours, corresponding to the hardware description languages SystemVerilog, Verilog, VHDL and GDL. These flavours provide a syntax for PSL that is similar to the syntax of the corresponding hardware description language. This enables hardware designers to specify and document their hardware-designs in a syntax that they are familiar with.

In this work, only the Boolean and temporal layers will be considered. Furthermore,

mainly the formal syntax of PSL is used, which differs from the syntax of all four layers. However, some operators are denoted differently from the formal syntax in this work to avoid problems with LTL operators that have the same syntax but different semantics.

The Boolean layer essentially consists of propositional logic. As mentioned above, the temporal layer is divided into FL and OBE. FL is a linear temporal logic that contains:

- propositional operators
- future temporal (LTL) operators
- a clocking operator for defining the granularity of time, which may differ from one part of a formula to another
- Sequential Extended Regular Expressions (SEREs), for defining finite-length regular patterns, together with strong and weak promotions of SEREs to formulas and an implication operator for predicating a formula on match of the pattern specified by a SERE
- an operator for aborting a formula 'asynchronously' on satisfaction of a propositional condition

Additionally, FL contains a lot of syntactic sugar, which is omitted here. Clocked statements may be seen as syntactic sugar, too, because formulas with clock statements can be easily rewritten to unclocked formulas. The necessary rewrite rules are even given in the official language standard [2].

OBE is essentially the temporal logic CTL [21], which is widely used and well understood. In this work, only FL is considered. Therefore, only this subset of PSL is formally introduced here.

The formal semantics of PSL uses two special states $\top$ and $\bot$. The state $\top$ satisfies every propositional formula, even the formula false. On the other hand, $\bot$ satisfies no propositional formula, even the formula true is not satisfied. Using these two special states, the semantics of a propositional formula $\varphi \in \mathsf{prop}_{\mathcal{V}}$ with respect to a state $s \in \mathcal{P}(\mathcal{V}) \cup \{\top, \bot\}$ is given by

- $\top \models_{\mathsf{xprop}} \varphi$
- $\bot \not\models_{\mathsf{xprop}} \varphi$
- $s' \models_{\mathsf{xprop}} \varphi$ iff $s' \models_{\mathsf{prop}} \varphi$ for $s' \in \mathcal{P}(\mathcal{V})$, i. e. for $s' \notin \{\top, \bot\}$

For a given set of variables $\mathcal{V}$, the set of *extended states over* $\mathcal{V}$ is denoted by $\mathcal{XP}(\mathcal{V}) := \mathcal{P}(\mathcal{V}) \cup \{\top, \bot\}$. The definition of the formal syntax of PSL uses a special function for words over these extended states. For finite or infinite words $w \in \mathcal{XP}(\mathcal{V})^\omega \cup \mathcal{XP}(\mathcal{V})^*$,

the word $\overline{w}$ denotes the word over states that is obtained from w by replacing every $\top$ with $\bot$ and vice versa, i. e. for all $i < |w|$, the following holds:

$$\overline{w}^i := \begin{cases} \bot & \text{if } w^i = \top \\ \top & \text{if } w^i = \bot \\ w^i & \text{otherwise} \end{cases}$$

Using these extended states and words over these states, it is possible to define the formal syntax and semantics of FL:

Definition 2.3.1 (Syntax of Sequential Extended Regular Expressions)
The following mutually recursive cases introduce the set of Sequential Extended Regular Expressions $\mathsf{sere}_\mathcal{V}$ *over a given set of variables* $\mathcal{V}$*:*

- *each propositional formula* $p \in \mathsf{prop}_\mathcal{V}$ *is a* SERE *over* $\mathcal{V}$
- $\{r\} \in \mathsf{sere}_\mathcal{V}$, *if* $r \in \mathsf{sere}_\mathcal{V}$
- $[*0] \in \mathsf{sere}_\mathcal{V}$
- $r[*] \in \mathsf{sere}_\mathcal{V}$, *if* $r \in \mathsf{sere}_\mathcal{V}$
- $r@c \in \mathsf{sere}_\mathcal{V}$, *if* $r \in \mathsf{sere}_\mathcal{V}$ *and* $c \in \mathsf{prop}_\mathcal{V}$
- $r_1 \,;\, r_2 \in \mathsf{sere}_\mathcal{V}$, *if* $r_1, r_2 \in \mathsf{sere}_\mathcal{V}$
- $r_1 : r_2 \in \mathsf{sere}_\mathcal{V}$, *if* $r_1, r_2 \in \mathsf{sere}_\mathcal{V}$
- $r_1 \mid r_2 \in \mathsf{sere}_\mathcal{V}$, *if* $r_1, r_2 \in \mathsf{sere}_\mathcal{V}$
- $r_1 \,\&\&\, r_2 \in \mathsf{sere}_\mathcal{V}$, *if* $r_1, r_2 \in \mathsf{sere}_\mathcal{V}$

Definition 2.3.2 (Syntax of Foundation Language (FL))
The following mutually recursive definitions introduce the set of FL *formulas* $\mathsf{fl}_\mathcal{V}$ *over a given set of variables* $\mathcal{V}$*:*

- $p, p! \in \mathsf{fl}_\mathcal{V}$, *if* $p \in \mathsf{prop}_\mathcal{V}$
- $\neg\varphi \in \mathsf{fl}_\mathcal{V}$, *if* $\varphi \in \mathsf{fl}_\mathcal{V}$
- $\varphi \wedge \psi \in \mathsf{fl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{fl}_\mathcal{V}$
- $r, r! \in \mathsf{fl}_\mathcal{V}$, *if* $r \in \mathsf{sere}_\mathcal{V}$
- $\underline{\mathsf{X}}\varphi$[6], $\varphi \,\underline{\mathsf{U}}\, \psi$[7] $\in \mathsf{fl}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{fl}_\mathcal{V}$

[6] written as $\mathsf{X!}\ \varphi$ in [2]
[7] written as $\varphi\ \mathsf{U}\ \psi$ in [2], but this operator corresponds to the $\underline{\mathsf{U}}$ operator of RLTL and LTL

- φ ABORT $b \in \mathsf{fl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{fl}_{\mathcal{V}}$, $b \in \mathsf{prop}_{\mathcal{V}}$
- $r \mapsto \varphi \in \mathsf{fl}_{\mathcal{V}}$, *if* $r \in \mathsf{sere}_{\mathcal{V}}$, $\varphi \in \mathsf{fl}_{\mathcal{V}}$
- $\varphi @ c \in \mathsf{fl}_{\mathcal{V}}$, *if* $\varphi \in \mathsf{fl}_{\mathcal{V}}$, $c \in \mathsf{prop}_{\mathcal{V}}$

Definition 2.3.3 (Important Subsets of FL)
The operator @ is called clock *operator. Therefore,* SEREs *without this clock operator are called* unclocked SEREs. *The set of all unclocked* SEREs *over a set of variables $\mathcal{V}$ is denoted by* $\mathsf{usere}_{\mathcal{V}}$. *As well,* FL *formulas without the clock operator are called* unclocked FL formulas. *The set of all unclocked* FL *formulas over $\mathcal{V}$ is denoted by* $\mathsf{ufl}_{\mathcal{V}}$. *The set of all* SERE*-free, unclocked* FL *formulas over $\mathcal{V}$ is denoted by* $\mathsf{sufl}_{\mathcal{V}}$.

There are two equivalent ways to define the formal semantics of FL. The so-called *clocked semantics* directly defines the semantics of SEREs and FL formulas. The so-called *unclocked semantics* reduces clocked SEREs and clocked FL formulas to unclocked ones, and then defines the semantics of unclocked SEREs and unclocked FL formulas. Both semantics are defined in the official language standard [2]. A proof of their equivalence can be found in Mike Gordon's embedding of PSL in HOL [26]. In this work, the unclocked semantics are used. The rewrite rules used to define this semantics use some syntactic sugar of FL formulas. Thus, some syntactic sugar has to be considered before:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- $\mathsf{X}\varphi := \neg\underline{\mathsf{X}}\neg\varphi$
- $\mathsf{F}\varphi := \mathsf{true}\ \underline{\mathsf{U}}\ \varphi$
- $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$
- $\varphi\ \mathsf{U}\ \psi$[8] $:= \varphi\ \underline{\mathsf{U}}\ \psi\ \vee\ \mathsf{G}\varphi$
- $\varphi\ \mathsf{B}\ \psi$[9] $:= \neg(\neg\varphi\ \underline{\mathsf{U}}\ \psi)$

Definition 2.3.4 (Semantics of unclocked SEREs)
The semantics of unclocked SEREs *is defined over finite words over extended states. For a finite word $v \in \mathcal{XP}(\mathcal{V})^*$ and an unclocked* SERE *r the notation $v \models_{\mathsf{usere}} r$ means that v* models *r* tightly. *For $r, r_1, r_2 \in \mathsf{usere}_{\mathcal{V}}$ and $b \in \mathsf{prop}_{\mathcal{V}}$, it is defined by:*

[8] written as [φ W ψ] in [2], but this operator corresponds to the U operator of RLTL and LTL
[9] written as [φ BEFORE!_ ψ] in [2], but this operator corresponds to the B operator of RLTL and LTL

- $v \models_{\mathsf{usere}} \{r\}$ *iff* $v \models_{\mathsf{usere}} r$
- $v \models_{\mathsf{usere}} b$ *iff* $|v| = 1$ *and* $v^0 \models_{\mathsf{xprop}} b$
- $v \models_{\mathsf{usere}} [*0]$ *iff* $v = \varepsilon$
- $v \models_{\mathsf{usere}} r[*]$ *iff either* $v \models_{\mathsf{usere}} [*0]$ *or* $\exists v_1, v_2$ *s. t.* $v_1 \neq \varepsilon$, $v = v_1 v_2$, $v_1 \models_{\mathsf{usere}} r$ *and* $v_2 \models_{\mathsf{usere}} r[*]$
- $v \models_{\mathsf{usere}} r_1 ; r_2$ *iff* $\exists v_1, v_2$ *s. t.* $v = v_1 v_2$, $v_1 \models_{\mathsf{usere}} r_1$ *and* $v_2 \models_{\mathsf{usere}} r_2$
- $v \models_{\mathsf{usere}} r_1 : r_2$ *iff* $\exists v_1, v_2, l$ *with* $|l| = 1$ *s. t.* $v = v_1 l v_2$, $v_1 l \models_{\mathsf{usere}} r_1$ *and* $l v_2 \models_{\mathsf{usere}} r_2$
- $v \models_{\mathsf{usere}} r_1 \mid r_2$ *iff* $v \models_{\mathsf{usere}} r_1$ *or* $v \models_{\mathsf{usere}} r_2$
- $v \models_{\mathsf{usere}} r_1 \,\&\&\, r_2$ *iff* $v \models_{\mathsf{usere}} r_1$ *and* $v \models_{\mathsf{usere}} r_2$

SEREs are an extended variant of regular expressions as introduced by Kleene [33]. They are used to describe properties of finite words. The length of a word may be considered. In particular, a SERE may model a prefix of a word tightly, while it does not model the word itself tightly. To check whether a prefix of a word matches some pattern described by a SERE, operators of FL can be used.

The most basic SEREs are of the form $[*0]$ and b. The SERE $[*0]$ models exactly the empty word ε tightly. b models all words of length one (i. e. all states) tightly that model the propositional formula b. Two SEREs r_1 and r_2 can be concatenated by the operators ; and :. The SERE $r_1 : r_2$ models all words v tightly that are concatenations of two words v_1 and v_2 such that r_1 models v_1 and r_2 models v_2 tightly. For example, the SERE $p_1 ; p_2$ with $p_1, p_2 \in \mathsf{prop}_{\mathcal{V}}$ models a word v tightly iff $|v| = 2$, $v^0 \models_{\mathsf{prop}} p_1$ and $v^1 \models_{\mathsf{prop}} p_2$ hold. The semantics of $r_1 : r_2$ is similar to the semantics of $r_1 ; r_2$, however it requires that v_1 and v_2 overlap by exactly one letter. $r[*]$ is used to describe a concatenation of a finite list of words $v_1, \ldots, v_n$ that are all modelled tightly by r. Finally, | and && describe disjunction and conjunction for SEREs. However, notice that there is no negation operator.

Definition 2.3.5 (Semantics of unclocked FL)
For propositional formulas $b, c \in \mathsf{prop}_{\mathcal{V}}$*, an unclocked* SERE $r \in \mathsf{usere}_{\mathcal{V}}$ *and unlocked* FL *formulas* $\varphi, \psi \in \mathsf{ufl}_{\mathcal{V}}$*, the semantics of unclocked* FL *with respect to a finite or infinite word* $v \in \mathcal{XP}(\mathcal{V})^* \cup \mathcal{XP}(\mathcal{V})^\omega$ *is given by:*

- $v \models_{\mathsf{ufl}} b$ *iff* $|v| = 0$ *or* $v^0 \models_{\mathsf{xprop}} b$
- $v \models_{\mathsf{ufl}} b!$ *iff* $|v| > 0$ *and* $v^0 \models_{\mathsf{xprop}} b$
- $v \models_{\mathsf{ufl}} r!$ *iff* $\exists j.\, j < |v|$ *s. t.* $v^{0..j} \models_{\mathsf{usere}} r$

- $v \models_{\mathsf{ufl}} r$ *iff* $\exists j.\, j < |v|$ *s. t.* $v^{0..j} \top^\omega \models_{\mathsf{ufl}} r!$
- $v \models_{\mathsf{ufl}} \neg\varphi$ *iff* $\overline{v} \not\models_{\mathsf{ufl}} \varphi$
- $v \models_{\mathsf{ufl}} \varphi \wedge \psi$ *iff* $v \models_{\mathsf{ufl}} \varphi$ *and* $v \models_{\mathsf{ufl}} \psi$
- $v \models_{\mathsf{ufl}} \underline{\mathsf{X}}\varphi$ *iff* $|v| > 1$ *and* $v^{1..} \models_{\mathsf{ufl}} \varphi$
- $v \models_{\mathsf{ufl}} \varphi\, \underline{\mathsf{U}}\, \psi$ *iff* $\exists k.\, k < |v|$ *s.t.* $v^{k..} \models_{\mathsf{ufl}} \psi$ *and* $\forall j < k.\, v^{j..} \models \varphi$
- $v \models_{\mathsf{ufl}} \varphi\, \mathsf{ABORT}\, b$ *iff either* $v \models_{\mathsf{ufl}} \varphi$ *or* $\exists j.j < |v|$ *s.t.* $v^j \models_{\mathsf{ufl}} b$ *and* $v^{0..j-1}\top^\omega \models_{\mathsf{ufl}} \varphi$
- $v \models_{\mathsf{ufl}} r \mapsto \varphi$ *iff* $\forall j < |v|.\, \overline{v}^{0..j} \models_{\mathsf{usere}} r \Longrightarrow v^{j..} \models_{\mathsf{ufl}} \varphi$

A word v is said to model *(or* to satisfy*) an unclocked* FL *formula φ iff $v \models_{\mathsf{ufl}} \varphi$ holds. An unclocked* FL *formula φ is* equivalent *to an unclocked* FL *formula ψ (denoted by $\varphi \equiv_{\mathsf{ufl}} \psi$) if for all words v, the relation $v \models_{\mathsf{ufl}} \varphi$ holds iff $v \models_{\mathsf{ufl}} \psi$ holds.*

There are two groups of unclocked FL operators: operators that correspond to RLTL operators and operators that are related to SEREs. The operators related to SEREs will be explained first: For a given SERE r, the FL formula $r!$ checks whether a prefix of a word v is modelled tightly by r. A weak variant of this formula is the formula r. It means that there is an extension of v such that an prefix of this extension is tightly modelled by the SERE r. The so-called *suffix implication* operator $\mapsto$ remains. For a given word v the formula $r \mapsto \varphi$ means that if some prefix of v models r tightly, then the remaining suffix (included the last letter of the prefix) models φ.

All other FL operators correspond to RLTL operators. A difference to RLTL is, that unclocked FL is able to consider finite paths. Thus, for a propositional formula b a strong variant $b!$ is introduced that does not accept the empty word ε. Analogously, $\underline{\mathsf{X}}$ is introduced as a strong variant of X. The semantics of $\underline{\mathsf{X}}$ requires that a next state exists. In contrast, the weak variants $\mathsf{X}\varphi$ and b accept ε and words shorter than two letters, respectively. For the remaining temporal operator $\underline{\mathsf{U}}$ a weak variant U is already present in RLTL. Apart from finite paths, the meaning of the FL operators is the same as the meaning of the corresponding RLTL operators. The role of the two special states $\top, \bot$ is played by the acceptance / rejection conditions of RLTL. The proof of this connection between PSL and RLTL is one important part of the translation presented in this work and will be explained in Section 3.1.

Using these semantics of unclocked SEREs and unclocked FL, the semantics of clocked SEREs and clocked FL are defined as follows:

Definition 2.3.6 (Semantics of clocked SEREs**)**
The semantics of clocked SEREs *is defined by reducing clocked* SEREs *to unclocked* SEREs*. For a finite word $v \in \mathcal{XP}(\mathcal{V})^*$, a clock $c \in \mathsf{prop}_{\mathcal{V}}$ and a* SERE *r, the notation $v \models^c_{\mathsf{sere}} r$ means that v* models *r* tightly in context of clock *c. This relation is defined by $v \models^c_{\mathsf{sere}} r :\Leftrightarrow v \models_{\mathsf{usere}} \mathcal{R}^c(r)$, in which the rewrite relation $\mathcal{R}^c$ is given by:*

- $\mathcal{R}^c(\{r\}) := \mathcal{R}^c(r)$
- $\mathcal{R}^c(b) := \neg c[*] \,;\, c \wedge b$
- $\mathcal{R}^c([*0]) := [*0]$
- $\mathcal{R}^c(r[*]) := \{\mathcal{R}^c(r)\}[*]$
- $\mathcal{R}^c(r@c_1) := \mathcal{R}^{c_1}(r)$
- $\mathcal{R}^c(r_1 \,;\, r_2) := \mathcal{R}^c(r_1) \,;\, \mathcal{R}^c(r_2)$
- $\mathcal{R}^c(r_1 : r_2) := \mathcal{R}^c(r_1) : \mathcal{R}^c(r_2)$
- $\mathcal{R}^c(r_1 \mid r_2) := \mathcal{R}^c(r_1) \mid \mathcal{R}^c(r_2)$
- $\mathcal{R}^c(r_1 \,\&\&\, r_2) := \mathcal{R}^c(r_1) \,\&\&\, \mathcal{R}^c(r_2)$

Definition 2.3.7 (Semantics of clocked FL)
The semantics of clocked FL *is defined by reducing clocked* FL *formulas to unclocked ones. For a finite or infinite word* $v \in \mathcal{XP}(\mathcal{V})^* \cup \mathcal{XP}(\mathcal{V})^\omega$*, a clock* $c \in \mathsf{prop}_\mathcal{V}$ *and a* FL *formula* φ*, the notation* $v \models^c_{\mathsf{fl}} \varphi$ *means that* v models (or satisfies) φ in context of clock c*. This relation is defined by* $v \models^c_{\mathsf{fl}} \varphi :\Leftrightarrow v \models_{\mathsf{ufl}} \mathcal{F}^c(\varphi)$*, in which the rewrite relation* $\mathcal{F}^c$ *is given by:*

- $\mathcal{F}^c(b) := \neg c \,\mathsf{U}\, c \wedge b$
- $\mathcal{F}^c(b!) := \neg c \,\underline{\mathsf{U}}\, c \wedge b$
- $\mathcal{F}^c(r) := \mathcal{R}^c(r)$
- $\mathcal{F}^c(r!) := \mathcal{R}^c(r)!$
- $\mathcal{F}^c(\neg\varphi) := \neg\mathcal{F}^c(\varphi)$
- $\mathcal{F}^c(\varphi \wedge \psi) := \mathcal{F}^c(\varphi) \wedge \mathcal{F}^c(\psi)$
- $\mathcal{F}^c(\underline{\mathsf{X}}\varphi) := \neg c \,\underline{\mathsf{U}}\, \big(c \wedge \underline{\mathsf{X}}\neg c \,\underline{\mathsf{U}}\, (c \wedge \mathcal{F}^c(\varphi))\big)$
- $\mathcal{F}^c(\varphi \,\underline{\mathsf{U}}\, \psi) := c \rightarrow \mathcal{F}^c(\varphi) \,\underline{\mathsf{U}}\, c \wedge \mathcal{F}^c(\psi)$
- $\mathcal{F}^c(\varphi \,\mathsf{ABORT}\, b) := \mathcal{R}^c(\varphi) \,\mathsf{ABORT}\, b$
- $\mathcal{F}^c(\varphi@c_1) := \mathcal{F}^{c_1}(\varphi)$
- $\mathcal{F}^c(r \mapsto \varphi) := \mathcal{R}^c(r) \mapsto \mathcal{F}^c(\varphi)$

Clocks are used to express different granularities of time. Informally, all points of time at which the clock does not hold, are ignored. The semantics of the ABORT operator are an exception. Even if the clock is not present, an occurrence of the acceptance condition is respected.

2.4 ω-Automata

ω-automata have been introduced by J. R. Büchi in 1960 [14]. They are similar to finite state automata as introduced by Kleene in 1956 [33]. While finite state automata decide whether a finite word belongs to some language, ω-automata decide this property for infinite words.

2.4.1 Finite State Automata on Finite Words

Definition 2.4.1 (Semiautomata)
A semiautomaton $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ *is a tuple where* $\mathcal{S}$ *is the finite set of* states, Σ *a finite alphabet,* $\mathcal{I} \subseteq \mathcal{S}$ *is the set of* initial states *and* $\mathcal{R} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ *is the* transition relation *of* $\mathfrak{A}$*. If for every state* $s \in \mathcal{S}$ *and every input* $i \in \Sigma$*, at least one* $s' \in \mathcal{S}$ *with* $(s, i, s') \in \mathcal{R}$ *exists, the semiautomaton* $\mathfrak{A}$ *is called* total*. If for every* s, i *at most one state* s' *with this property exists,* $\mathfrak{A}$ *is called* deterministic*.*

Definition 2.4.2 (Run of a Word)
Given a semiautomaton $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ *and a finite or infinite word* α *over* Σ*. Then, each word* β *over* $\mathcal{S}$ *with*

- β *is infinite if* α *is infinite and* $|\beta| = |\alpha| + 1$ *if* α *is finite*
- $\beta^0 \in \mathcal{I}$
- $(\beta^i, \alpha^i, \beta^{i+1}) \in \mathcal{R}$ *for all* $i < |\alpha|$

is called a run of α through $\mathfrak{A}$*. The set of all runs of a word* α *through a semiautomaton* $\mathfrak{A}$ *is denoted by* $\mathsf{RUN}_{\mathfrak{A}}(\alpha)$*.*

Example 2.4.3 *Let* $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ *be a semiautomaton (see Figure 2.1) with:*

- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\mathcal{S} = \{s_0, s_1, s_2\}$
- $\mathcal{I} = \{s_0\}$
- $\mathcal{R} = \{(s_i, j, s_k) \mid i + j \equiv k \mod 3\}$

Then, $s_0 s_0 s_1 s_0$ *is a run of the finite word* $\alpha = 312$ *over* Σ *through* $\mathfrak{A}$*. This is the only run of* α *through* $\mathfrak{A}$*, since* $\mathfrak{A}$ *is total and deterministic and therefore has exactly one run for every word.*

Definition 2.4.4 (Finite Automata on Finite Words)
A finite automaton on finite words is a tuple $\mathfrak{A} = (\mathfrak{B}, \mathcal{F})$ *where* $\mathfrak{B} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ *is a semiautomaton and* $\mathcal{F} \subseteq \mathcal{S}$ *a set of* final states*.* $\mathfrak{A}$ accepts a finite word $\alpha \in \Sigma^*$ *iff*

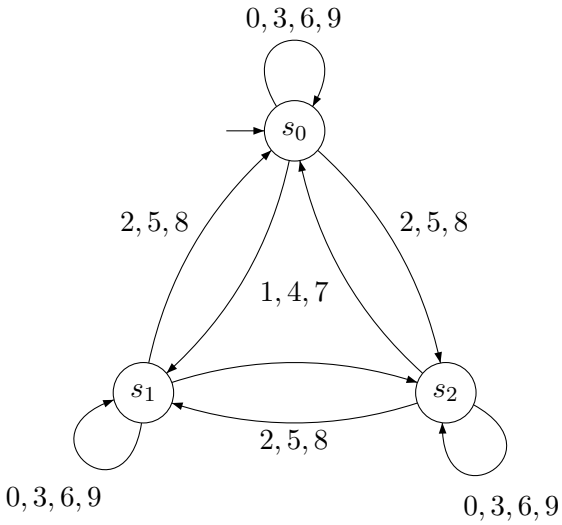

Figure 2.1: Example 2.4.3

there is a run $\beta \in \mathsf{RUN}_{\mathfrak{A}}(\alpha)$ *that ends in a final state, i. e. iff there is a run* β *with* $\beta^{|\beta|-1} \in \mathcal{F}$.

A finite automaton over finite words $\mathfrak{A}$ *is often denoted by* $(\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{F})$. *A finite automaton is called* total *or* deterministic *if the corresponding semiautomaton has these properties.*

Example 2.4.5 *Let* $\mathfrak{A}$ *be the extension of the semiautomaton from Example 2.4.3 to a finite automaton on finite words with* $\mathcal{F} = \{s_0\}$. *Then,* $\mathfrak{A}$ *accepts exactly the empty word* ε *and every decimal number that is divisible by three.*

2.4.2 ω-Automata

ω-automata are finite automata on infinite words. Similarly to the case of finite words, a set of accepting states is often used to define the acceptance condition. However, in the case of infinite words, there are several reasonable definitions of acceptance conditions. For example, an infinite word α could be accepted by an automaton $\mathfrak{A}$ iff there exists a run of α through $\mathfrak{A}$ that

- never leaves the set of accepting states
- visits the set of accepting states at least once
- from some point of time never leaves the set of accepting states
- visits the set of accepting states infinitely often.

The last acceptance condition is the one used by Büchi. Therefore, the resulting ω-automata are called Büchi automata. However, the other acceptance conditions

and a lot of similar ones are used in practice, too. They lead to different classes of ω-automata with different expressive power. In this work, a symbolic representation of ω-automata is used, so-called *automaton formulas*. This symbolic representation is able to express all presented acceptance conditions.

2.4.3 Symbolic Representation

Although good model checking procedures for CTL were known [19], first implementations of these procedures were not able to verify large systems, because no efficient data structures were used. Verification tools were only able to handle systems with a thousand states. A breakthrough was achieved by representing the systems with Boolean functions, which are stored as *binary decision diagrams (BDDs)* [9]. The resulting *symbolic model checking* procedures [8, 11, 12, 13] allow the checking of systems with more than 10^{20} states.

Symbolic representations of ω-automata have many advantages: In general, the symbolic representation of an ω-automaton is exponentially more succinct than the corresponding ω-automaton. This allows linear translation procedures of LTL to symbolically represented ω-automata [48]. Moreover, the resulting ω-automata can directly be used for symbolic model checking. Additionally, symbolically represented ω-automata can be handled like formulas of a logic [48].

To explain the symbolic representation of ω-automata, semiautomata are considered first: Let $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ be a semiautomaton. As the set Σ and the set of states $\mathcal{S}$ are finite, they can be encoded by a finite set of propositional variables. So, let $\Sigma = \mathcal{P}(\mathcal{V}_\Sigma)$ with $\mathcal{V}_\Sigma = \{i_0, \ldots, i_n\}$ and $\mathcal{S} = \mathcal{P}(\mathcal{V}_\mathcal{S})$ with $\mathcal{V}_\mathcal{S} = \{q_0, \ldots, q_m\}$ hold. With these settings, a state of $\mathfrak{A}$ is a subset of $\mathcal{V}_\mathcal{S}$, and a letter of the input alphabet Σ is a subset of $\mathcal{V}_\Sigma$. Those subsets of a set of propositional variables can be interpreted as assignments. Therefore, it is possible to encode a set of those subsets S, for example the set of initial states $\mathcal{I}$, by a propositional formula Φ_S that has the following property: $s \in S \Leftrightarrow s \models_{\mathsf{prop}} \Phi_S$.

Notice, that with n propositional variables 2^n states can be encoded. Moreover, small propositional formulas can encode large sets. For example, if every state of the semiautomaton is an initial state, the set of initial states $\mathcal{I}$ can be encoded by the propositional formula true.

Example 2.4.6 *Let $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ be the semiautomaton from Example 2.4.3, i. e. the semiautomaton given by:*

- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\mathcal{S} = \{s_0, s_1, s_2\}$
- $\mathcal{I} = \{s_0\}$
- $\mathcal{R} = \{(s_i, j, s_k) \mid i + j \equiv k \mod 3\}$

Then, the ten letters of Σ can be encoded by four propositional variables i_0, i_1, i_2 and i_3. For this purpose, a kind of binary encoding is used: 0 is encoded by $\emptyset$ (short $0 \mathrel{\hat{=}} \emptyset$), $1 \mathrel{\hat{=}} \{i_0\}$, $2 \mathrel{\hat{=}} \{i_1\}$, $3 \mathrel{\hat{=}} \{i_1, i_0\}$, ... The set $\mathcal{S}$ can be encoded by two variables q_0 and q_1: $s_0 \mathrel{\hat{=}} \emptyset$, $s_1 \mathrel{\hat{=}} \{q_0\}$ and $s_2 \mathrel{\hat{=}} \{q_1\}$. With these settings, the set of initial states can be encoded by the formula $\neg q_0 \neg q_1$.

By encoding Σ and $\mathcal{S}$ by propositional variables additional states and inputs are introduced. For example, the state $\{q_0, q_1\}$ has not existed before. However, these additional states and inputs will not influence the semantics of an automaton, provided that no additional accepted runs are introduced. This can be easily achieved by a suitable transition relation.

The transition relation $\mathcal{R}$ can be encoded by a propositional formula, as well. $\mathcal{R}$ is a subset of $\mathcal{P}(\mathcal{V}_\mathcal{S}) \times \mathcal{P}(\mathcal{V}_\Sigma) \times \mathcal{P}(\mathcal{V}_\mathcal{S})$. As on has to distinguish between the two subsets of $\mathcal{P}(\mathcal{V}_\mathcal{S})$, for every state variable $q \in \mathcal{V}_\mathcal{S}$, a new state variable is introduced. These new state variables are used to describe the second subset of $\mathcal{P}(\mathcal{V}_\mathcal{S})$. Thus, the new state variable corresponding to a variable $q \in \mathcal{V}_\mathcal{S}$ represents the value of the variable q at the next point of time. Therefore, it is denoted by $\mathtt{X}q$. Notice that $\mathtt{X}q$ is a variable, while $\mathsf{X}q$ is a LTL formula. However, the meaning of the variable and the formula are similar. Using this additional set of state variables, the transition relation can be encoded by a propositional formula $\Phi_\mathcal{R}$ that fulfils the following property: for all states $s_1, s_2 \subseteq \mathcal{V}_\mathcal{S}$ and all inputs $\sigma \subseteq \mathcal{V}_\Sigma$ the relation $(s_1, \sigma, s_2) \in \mathcal{R} \Leftrightarrow \big(s_1 \cup \sigma \cup \{\mathtt{X}q \mid q \in s_2\}\big) \models_{\mathsf{prop}} \Phi_\mathcal{R}$ holds.

Example 2.4.7 *Let $\mathfrak{A} = (\Sigma, \mathcal{S}, \mathcal{I}, \mathcal{R})$ be the semiautomaton from Example 2.4.6 with the encodings $\mathcal{V}_\Sigma = \{i_0, i_1, i_2, i_3\}$, $\mathcal{S} = \{q_0, q_1\}$ and $\Phi_\mathcal{I} = \neg q_0 \neg q_1$. To encode the transition relation $\mathcal{R}$ by a formula $\Phi_\mathcal{R}$, the subsets of Σ, which are used to label the transitions (see Figure 2.2) are encoded first:*

- *$\{0, 3, 6, 9\}$ is encoded by $\varphi_0 = \neg i_0 \neg i_1 \neg i_2 \neg i_3 \vee i_0 i_1 \neg i_2 \neg i_3 \vee \neg i_0 i_1 i_2 \neg i_3 \vee i_0 \neg i_1 \neg i_2 i_3$*
- *$\{1, 4, 7\}$ is encoded by $\varphi_1 = i_0 \neg i_1 \neg i_2 \neg i_3 \vee \neg i_0 \neg i_1 i_2 \neg i_3 \vee i_0 i_1 i_2 \neg i_3$*
- *$\{2, 5, 8\}$ is encoded by $\varphi_2 = \neg i_0 i_1 \neg i_2 \neg i_3 \vee i_0 \neg i_1 i_2 \neg i_3 \vee \neg i_0 \neg i_1 \neg i_2 i_3$*

Using these encodings, the transition relation can be encoded by:

$$
\begin{array}{lll}
s_0 \xrightarrow{1,4,7} s_1\colon & \neg q_0 \neg q_1 \wedge \varphi_1 \wedge \mathtt{X}q_0 \neg \mathtt{X}q_1 & \vee \\
s_0 \xrightarrow{2,5,8} s_2\colon & \neg q_0 \neg q_1 \wedge \varphi_2 \wedge \neg \mathtt{X}q_0\ \mathtt{X}q_1 & \vee \\
s_1 \xrightarrow{0,3,6,9} s_1\colon & q_0 \neg q_1 \wedge \varphi_0 \wedge \mathtt{X}q_0 \neg \mathtt{X}q_1 & \vee \\
s_1 \xrightarrow{1,4,7} s_2\colon & q_0 \neg q_1 \wedge \varphi_1 \wedge \neg \mathtt{X}q_0\ \mathtt{X}q_1 & \vee \\
s_1 \xrightarrow{2,5,8} s_0\colon & q_0 \neg q_1 \wedge \varphi_2 \wedge \neg \mathtt{X}q_0 \neg \mathtt{X}q_1 & \vee \\
s_2 \xrightarrow{0,3,6,9} s_2\colon & \neg q_0\ q_1 \wedge \varphi_0 \wedge \neg \mathtt{X}q_0\ \mathtt{X}q_1 & \vee \\
s_2 \xrightarrow{1,4,7} s_0\colon & \neg q_0\ q_1 \wedge \varphi_1 \wedge \neg \mathtt{X}q_0 \neg \mathtt{X}q_1 & \vee \\
s_2 \xrightarrow{2,5,8} s_1\colon & \neg q_0\ q_1 \wedge \varphi_2 \wedge \neg \mathtt{X}q_0\ \mathtt{X}q_1 &
\end{array}
$$

This formula has not been simplified to show the relation between the original translation relation and its encoding.

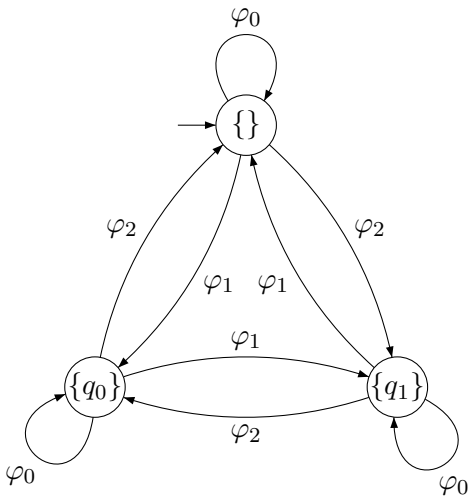

Figure 2.2: Example 2.4.6 and Example 2.4.7

2.4.4 Automaton Formulas

This symbolic representation of semiautomata directly leads to a symbolic representation of finite state automata over finite words, since the set of final states can be encoded in the same way. For ω-automata, this symbolic representation leads to *automaton formulas* [48, 49], that are defined as follows:

Definition 2.4.8 (Syntax of Flat Acceptance Conditions)
The following mutually recursive definitions introduce the set of flat acceptance conditions $\mathsf{ac}_\mathcal{V}$ *over a set of variables* $\mathcal{V}$*:*

- *every propositional formula* $p \in \mathsf{prop}_\mathcal{V}$ *is an acceptance condition over* $\mathcal{V}$
- $\neg\varphi \in \mathsf{ac}_\mathcal{V}$, *if* $\varphi \in \mathsf{ac}_\mathcal{V}$
- $\varphi \wedge \psi \in \mathsf{ac}_\mathcal{V}$, *if* $\varphi, \psi \in \mathsf{ac}_\mathcal{V}$
- $\mathsf{G}\varphi \in \mathsf{ac}_\mathcal{V}$, *if* $\varphi \in \mathsf{ac}_\mathcal{V}$

Definition 2.4.9 (Syntax of Automaton Formulas)
The following mutually recursive definitions introduce the set of automaton formulas $\mathcal{L}_\omega(\mathcal{V})$ *over a set of variables* $\mathcal{V}$*:*

- *every flat acceptance condition* $\Phi_\mathcal{F} \in \mathsf{ac}_\mathcal{V}$ *is an automaton formula over* $\mathcal{V}$

- $\neg\varphi \in \mathcal{L}_\omega(\mathcal{V})$, *if* $\varphi \in \mathcal{L}_\omega(\mathcal{V})$
- $\varphi \wedge \psi \in \mathcal{L}_\omega(\mathcal{V})$, *if* $\varphi, \psi \in \mathcal{L}_\omega(\mathcal{V})$
- $\mathcal{A}_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F}) \in \mathcal{L}_\omega$, *if* $\Phi_\mathcal{F} \in \mathcal{L}_\omega(Q)$ *and* $Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}$ *are the symbolic representations of the set of states, the set of initial states and the transition relation of a semiautomaton, i. e.* Q *is a set of variables with* $Q \cap \mathcal{V} = \emptyset$, $\Phi_\mathcal{I} \in \mathsf{prop}_Q$ *and* $\Phi_\mathcal{R} \in \mathsf{prop}_{Q \cup \mathcal{V} \cup \{\mathtt{x}q \mid q \in Q\}}$

Flat acceptance conditions are used to distinguish between the parts of an automaton formula that may contain automaton operators and the parts that may not contain these operators. This could as well be achieved without explicitly introducing flat acceptance conditions. However, the introduction of flat acceptance conditions is an appropriate way to model automaton formulas in HOL.

The most interesting operator is $\mathcal{A}_\exists$, which is called *existential automaton operator* in the following. A formula $\mathcal{A}_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F})$ consists of an acceptance condition $\Phi_\mathcal{F}$ and a symbolic representation $(\mathcal{V}_\Sigma, Q, \Phi_\mathcal{I}, \Phi_\mathcal{R})$ of a semiautomaton, whose input alphabet $\mathcal{V}_\Sigma$ is implicitly given by $\mathcal{V}$. The acceptance condition has to check whether an infinite run is accepted or not. Flat acceptance conditions, which are a simple subset of FutureLTL, may be used therefor. However, automaton formulas themselves are a formalism to describe languages of infinite words. Thus, also nested occurrences of automaton formulas are allowed. Notice, that is includes nested occurrences of automaton operators. This informal description of the semantics of automaton formulas leads to the following formal definition:

Definition 2.4.10 (Semantics of Flat Acceptance Conditions)
Flat acceptance conditions are a subset of LTL. *Therefore, the semantics of a flat acceptance condition is given by the semantics of* LTL. *However, as the operator* G *is defined as syntactic sugar in* LTL, *its semantics will be explain here: The semantics of a flat automaton formula* $\varphi \in \mathsf{ac}_\mathcal{V}$ *is for an infinite word* $v \in \mathcal{P}(\mathcal{V})^\omega$ *and a point of time* $t \in \mathbb{N}$ *given by*

- $v \models^t_{\mathsf{ac}} p$ *iff* $v^t \models_{\mathsf{prop}} p$
- $v \models^t_{\mathsf{ac}} \neg\varphi$ *iff* $v \not\models^t_{\mathsf{ac}} \varphi$
- $v \models^t_{\mathsf{ac}} \varphi \wedge \psi$ *iff* $v \models^t_{\mathsf{ac}} \varphi$ *and* $v \models^t_{\mathsf{ac}} \psi$
- $v \models^t_{\mathsf{ac}} \mathsf{G}\varphi$ *iff* $\forall k \geq t.\, v \models^k_{\mathsf{ac}} \varphi$

A word $v \in \mathcal{P}(\mathcal{V})^\omega$ *is said to satisfy a flat acceptance condition* φ *(short* $v \models_{\mathsf{ac}} \varphi$*) iff* $v \models^0_{\mathsf{ac}} \varphi$ *holds.*

Definition 2.4.11 (Semantics of Automaton Formulas)
The semantics of an automaton formula $\varphi \in \mathcal{L}_\omega(\mathcal{V})$ *is for an infinite word* $v \in \mathcal{P}(\mathcal{V})^\omega$ *given by:*

- $v \models_{\mathsf{omega}} \Phi_{\mathcal{F}}$ *iff* $v \models_{\mathsf{ac}} \Phi_{\mathcal{F}}$
- $v \models_{\mathsf{omega}} \neg\varphi$ *iff* $v \not\models_{\mathsf{omega}} \varphi$
- $v \models_{\mathsf{omega}} \varphi \wedge \psi$ *iff* $v \models_{\mathsf{omega}} \varphi$ *and* $v \models_{\mathsf{omega}} \psi$
- $v \models_{\mathsf{omega}} \mathcal{A}_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}})$ *iff an infinite word* $\beta \in Q^{\omega}$ *exists with*
  - $\beta^0 \models_{\mathsf{prop}} \Phi_{\mathcal{I}}$
  - $\left(\beta^i \cup v^i \cup \{\mathtt{X}q \mid q \in \beta^{i+1}\}\right) \models_{\mathsf{prop}} \Phi_{\mathcal{R}}$ *for all* $i \in \mathbb{N}$
  - $\beta \models_{\mathsf{omega}} \Phi_{\mathcal{F}}$

A word $v \in \mathcal{P}(\mathcal{V})^{\omega}$ *is said to satisfy an automaton formula* φ *iff* $v \models_{\mathsf{omega}} \varphi$ *holds. An automaton formula* φ *is* equivalent *to an automaton formula* ψ *(denoted by* $\varphi \equiv_{\mathsf{omega}} \psi$*) iff for all* v *the relation* $v \models_{\mathsf{omega}} \varphi$ *holds iff* $v \models_{\mathsf{omega}} \psi$ *holds.*

2.4.5 Syntactic Sugar

Automaton formulas are able to express all ω-automata classes mentioned before. However, to be able to express these classes in a convenient way, some syntactic sugar for automaton formulas and flat acceptance conditions is needed:

- $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi := \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- $\mathsf{F}\varphi := \neg\mathsf{G}\neg\varphi$
- $\mathcal{A}_{\forall}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}}) := \neg\mathcal{A}_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \neg\Phi_{\mathcal{F}})$

In practice, another extension is useful, too. This extension is similar to syntactic sugar, as far as it does not increase the expressiveness of automaton formulas. However, it cannot be easily defined. Consider a formula of the form $\mathcal{A}_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}}) \in \mathcal{L}_{\omega}(\mathcal{V})$. In this formula, there is a distinction between the *input variables* $\mathcal{V}$ and the *state variables* Q. The formula $\Phi_{\mathcal{I}} \in \mathsf{prop}_Q$ may only consider state variables. The transition relation $\Phi_{\mathcal{R}}$ may use the input variables, the state variables and the state variables at the next point of time, but not the input variables at the next point of time. Finally the acceptance condition $\Phi_{\mathcal{F}}$ may only consider state variables.

Therefore, an useful extension is to weaken this distinction. This extension leads to a new operator $A_{\exists}$. This new operator is very similar to $\mathcal{A}_{\exists}$. However, only the following constraints are required for $A_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}}) \in \mathcal{L}_{\omega}(\mathcal{V})$:

- $\Phi_{\mathcal{F}} \in \mathcal{L}_{\omega}(Q \cup \mathcal{V})$,
- $Q \cap \mathcal{V} = \emptyset$,

- $\Phi_{\mathcal{I}} \in \mathsf{prop}_{Q \cup \mathcal{V}}$ and
- $\Phi_{\mathcal{R}} \in \mathsf{prop}_{Q \cup \mathcal{V} \cup \{\mathtt{X}q \mid q \in Q \cup \mathcal{I}\}}$.

The semantics is given by: $v \models_{\mathsf{omega}} A_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}})$ iff an infinite word $\beta \in Q^{\omega}$ exists with

- $\left(\beta^0 \cup v^0\right) \models_{\mathsf{prop}} \Phi_{\mathcal{I}}$
- $\left(\beta^i \cup v^i \cup \{\mathtt{X}q \mid q \in \beta^{i+1} \cup v^{i+1}\}\right) \models_{\mathsf{prop}} \Phi_{\mathcal{R}}$ for all $i \in \mathbb{N}$
- $\beta \cup v \models_{\mathsf{omega}} \Phi_{\mathcal{F}}$, where $\beta \cup v$ is the pointwise union of β and v, i. e. $(\beta \cup v)^i := \beta^i \cup v^i$ for all $i \in \mathbb{N}$.

This extended operator $A_{\exists}$ can be reduced to $\mathcal{A}_{\exists}$ by introducing new state variables and fixing the values of these new variables by extending the transition relation. For example, the formula $A_{\exists}(Q, \Phi_{\mathcal{I}} \wedge i_0, \Phi_{\mathcal{R}} \wedge \neg \mathtt{X} i_0, \Phi_{\mathcal{F}} \vee i_1) \in \mathcal{L}_{\omega}(\mathcal{V})$ with $i_0, i_1 \in \mathcal{V}$ can be reduced to $\mathcal{A}_{\exists}(Q \cup \{j_0, j_1\}, \Phi_{\mathcal{I}} \wedge j_0, \Phi_{\mathcal{R}} \wedge \neg \mathtt{X} j_0 \wedge (j_0 \leftrightarrow i_0) \wedge (j_1 \leftrightarrow i_1), \Phi_{\mathcal{F}} \vee j_1)$ for $j_0, j_1 \notin \mathcal{V} \cup Q$. Therefore, this new operator $A_{\exists}$ can be considered as syntactic sugar. It does not increase the expressiveness of automaton formulas. In fact, symbolic model checking algorithms can naturally handle these extended automaton formulas. Moreover, using the $A_{\exists}$ operator, automaton formulas may be written linearly more succinct.

Similar to the syntactic sugar defined with the help of the $\mathcal{A}_{\exists}$ operator, some syntactic sugar is introduced that uses the new operator $A_{\exists}$:

- $A_{\forall}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}}) := \neg A_{\exists}(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \neg \Phi_{\mathcal{F}})$
- $\hat{A}_{\exists}\left(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \mathcal{F}, \Phi_p\right) := A_{\exists}\left(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \left(\bigwedge_{\xi \in \mathcal{F}} \xi\right) \wedge \Phi_p\right)$
- $\hat{A}_{\forall}\left(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \mathcal{F}, \Phi_p\right) := A_{\forall}\left(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \left(\bigwedge_{\xi \in \mathcal{F}} \xi\right) \rightarrow \Phi_p\right)$

The operator $A_{\forall}$ is very similar to $\mathcal{A}_{\forall}$. The operators $\hat{A}_{\exists}$ and $\hat{A}_{\forall}$ are used by the translation of LTL to ω-automata. They allow one to split the acceptance condition $\Phi_{\mathcal{F}} \in \mathcal{L}_{\omega}(Q \cup \mathcal{V})$ into a propositional part $\Phi_p \in \mathsf{prop}_{Q \cup \mathcal{V}}$ and a finite set of constraints $\mathcal{F}$ with $\xi \in \mathcal{L}_{\omega}(Q \cup \mathcal{V})$ for all $\xi \in \mathcal{F}$.

In addition to weakening the usage of input variables, it is useful to weaken the usage of X. The expression $\mathtt{X}q$ is used by the transition relation to denote a special variable that corresponds to the value of the variable q at the next point of time. Sometimes, it is convenient to use a similar operator to consider the value of a propositional formula φ at the next point of time. Therefore, the operator **X** is introduced. It replaces every occurrence of a variable q in a propositional formula with $\mathtt{X}q$. Formally, the operator **X** is defined by:

- $\mathbf{X}(q) := \mathtt{X}q$ for all variables q
- $\mathbf{X}(\neg\varphi) := \neg\mathbf{X}(\varphi)$ and
- $\mathbf{X}(\varphi \wedge \psi) := \mathbf{X}(\varphi) \wedge \mathbf{X}(\psi)$

2.4.6 Flat Automaton Formulas

Automaton formulas are a convenient way to represent ω-automata. However, the connection between automaton formulas and ω-automata is not obvious in general, because some automaton formulas like $\mathcal{A}_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \neg\mathcal{A}_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}}))$ or $\mathcal{A}_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \wedge \mathcal{A}_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2})$ contain more than one automaton operator and other automaton formulas like $\mathsf{G}q$ contain no automaton operators.

On the other hand, automaton formulas of the form $\mathcal{A}_\exists(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}})$ where $\Phi_{\mathcal{F}}$ is a flat acceptance condition are obviously related to ω-automata. Those automaton formulas are called *flat*. Notice that following this definition $A_\exists(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}})$ and $\hat{A}_\exists(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \mathcal{F}, \Phi_p)$ are flat automaton formulas, too, if $\Phi_{\mathcal{F}} \in \mathsf{ac}$ and $\xi \in \mathsf{ac}$ for all $\xi \in \mathcal{F}$. Since a flat automaton formula φ directly corresponds to an ω-automaton $\mathfrak{A}_\varphi$, it is said that φ is *total* or *deterministic* iff $\mathfrak{A}_\varphi$ is total or deterministic.

For every automaton formula φ, there is a flat automaton formula φ_{flat} that is equivalent to φ [49]. The translation of an automaton formula to an equivalent flat one is called *flattening*. To flatten an automaton formula, formulas of the form given above have to be eliminated by introduction or elimination of automaton operators. It is quite easy to introduce an automaton operator, as the following lemma shows:

Lemma 2.4.12 *For all automaton formulas $\varphi \in \mathcal{L}_\omega(\mathcal{V})$, the following holds*[10]*:*

$$A_\exists(\{\}, \mathsf{true}, \mathsf{true}, \varphi) \equiv_{\mathsf{omega}} \varphi$$

However, it is much harder to eliminate automaton operators. Thereby, the *product* of automaton formulas is needed:

Definition 2.4.13 (Product of Automaton Formulas)
For two automaton formulas $\mathcal{A}_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1})$ and $\mathcal{A}_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2})$ their product *is defined by:*

$$\begin{gathered}\mathcal{A}_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \times \mathcal{A}_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2}) := \\ \mathcal{A}_\exists(Q_1 \cup Q_2, \Phi_{\mathcal{I}_1} \wedge \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_1} \wedge \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_1} \wedge \Phi_{\mathcal{F}_2})\end{gathered}$$

For $A_\exists$ and $\hat{A}_\exists$, this definition becomes:

$$\begin{gathered}A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \times A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2}) := \\ A_\exists(Q_1 \cup Q_2, \Phi_{\mathcal{I}_1} \wedge \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_1} \wedge \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_1} \wedge \Phi_{\mathcal{F}_2})\end{gathered}$$

$$\begin{gathered}\hat{A}_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \mathcal{F}_1, \Phi_{p_1}) \times \hat{A}_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \mathcal{F}_2, \Phi_{p_2}) := \\ \hat{A}_\exists(Q_1 \cup Q_2, \Phi_{\mathcal{I}_1} \wedge \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_1} \wedge \Phi_{\mathcal{R}_2}, \mathcal{F}_1 \cup \mathcal{F}_2, \Phi_{p_1} \wedge \Phi_{p_2})\end{gathered}$$

[10] theorem `ID_AUTOMATON_SEM` in theory `Omega_Automata_Lemmata`

The product of automaton formulas can be used to flatten automaton formulas, because the following lemmata hold:

Lemma 2.4.14 *For all automaton formulas* $A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}}) \in \mathcal{L}_\omega(\mathcal{V})$*, the following holds*[11]*:*

$$A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}})) \equiv_{\mathsf{omega}} A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \mathsf{true}) \times A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}})$$

Notice, that according to the definition of the syntax of automaton formulas, Q_1*,* Q_2 *and* $\mathcal{V}$ *are pairwise disjoint.*

Lemma 2.4.15 *For all* $A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \in \mathcal{L}_\omega(\mathcal{V})$ *and* $A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2}) \in \mathcal{L}_\omega(\mathcal{V})$ *with* $Q_1 \cap Q_2 = \emptyset$*, the following holds*[12]*:*

$$A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \wedge A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2}) \equiv_{\mathsf{omega}} A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1}) \times A_\exists(Q_2, \Phi_{\mathcal{I}_2}, \Phi_{\mathcal{R}_2}, \Phi_{\mathcal{F}_2})$$

To flatten an arbitrary automaton formula, it remains to translate an automaton formula φ of the form $\neg A_\exists(Q_1, \Phi_{\mathcal{I}_1}, \Phi_{\mathcal{R}_1}, \Phi_{\mathcal{F}_1})$ to a flat automaton formula. In general, this translation needs exponential time. It is usually done by determinising φ. The negation of a total, deterministic automaton formula $\varphi_{\mathsf{det,total}}$ can be easily computed by simply negating the acceptance condition of $\varphi_{\mathsf{det,total}}$[13]. Due to these exponential determinisation steps, the flattening of automaton formulas has nonelementary complexity [49].

2.4.7 Classes of ω-Automata

Flat automaton formulas of the form $\mathcal{A}_\exists(Q, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \mathsf{GF}\varphi)$, where φ is a propositional formula, correspond to Büchi automata. In fact, for every automaton formula, an equivalent formula of this form exists, since Büchi automata are expressively complete with respect to ω-regular properties. However, there are also a lot of other classes of ω-automata. One reason why other classes of ω-automata are interesting too is, that in contrast to classical finite state automata, Büchi automata are not closed under determinisation, i. e. not for every Büchi automaton $\mathfrak{A}$, a deterministic Büchi automaton $\mathfrak{B}$ exists that is equivalent to $\mathfrak{A}$. Examples of classes of ω-automata defined for this reason are *Rabin* [39, 43] and *Streett automata* [53]. Rabin and Streett automata are as expressive as Büchi automata, but they are closed under determinisation. Another reason to consider additional classes of ω-automata is, that the model-checking problem of

[11] theorem `A_NDET_EX__FLATTENING` in theory `Omega_Automata_Lemmata`
[12] theorem `A_AND__A_NDET_EX` in theory `Omega_Automata_Lemmata`
[13] theorem `TOTAL_DET_AUTOMATON_EX_ALL_EQUIV` in theory `Omega_Automata_Lemmata`

some strictly less expressive classes of ω-automata is in practice more efficiently solvable than the model-checking problem of Büchi automata. For example, deterministic *safety*, *liveness*, *prefix* and *persistence automata* can be translated to alternation free μ-calculus, while the translation of deterministic Büchi and Streett automata requires an alternation-depth of two in the translation [49]. Since the alternation-depth mainly determines the runtime of the verification of μ-calculus formulas [49], the classes of ω-automata that can be translated to alternation free μ-calculus can be handled much more efficiently in practice. However, safety, liveness and persistence automata are still expressive enough for a lot of practical applications. Hence, these automaton classes are particularly interesting for specification and verification.

The classes of ω-automata are defined by their acceptance condition. The most important classes of acceptance conditions are:

Definition 2.4.16 (Classes of Acceptance Conditions)
Let Φ_i, Ψ_i be propositional formulas for all $i \in \{0, \ldots, f\}$. Then, the following classes of acceptance conditions are defined [37]:

Safety condition:	$\mathsf{G}\Phi_0$
Liveness condition:	$\mathsf{F}\Phi_0$
Büchi condition [14, 15]:	$\mathsf{GF}\Phi_0$
Persistence condition [37]:	$\mathsf{FG}\Phi_0$
Rabin condition [44]:	$\bigvee_{j=0}^{f} \left(\mathsf{GF}\Phi_0 \wedge \mathsf{FG}\Psi_0\right)$
Streett condition [53]:	$\bigwedge_{j=0}^{f} \left(\mathsf{FG}\Phi_0 \vee \mathsf{GF}\Psi_0\right)$
Prefix condition (1. kind) [37, 47]:	$\bigwedge_{j=0}^{f} \left(\mathsf{G}\Phi_0 \vee \mathsf{F}\Psi_0\right)$
Prefix condition (2. kind) [37, 47]:	$\bigvee_{j=0}^{f} \left(\mathsf{F}\Phi_0 \vee \mathsf{G}\Psi_0\right)$

Definition 2.4.17 (Classes of ω-Automata)
The class of nondeterministic ω-automata corresponding to a class of acceptance conditions is given by the set of all flat automaton formulas $\mathcal{A}_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F})$, where $\Phi_\mathcal{F}$ belongs to the class of acceptance conditions. As well, $A_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F})$ can be used to define the classes of ω-automata. The class of (nondeterministic) safety, liveness, Büchi, persistence, Prefix-1, Prefix-2, Rabin and Streett automata are denoted by NDET_G, NDET_F, $\mathsf{NDET}_\mathsf{GF}$, $\mathsf{NDET}_\mathsf{FG}$, $\mathsf{NDET}_\mathsf{Prefix1}$, $\mathsf{NDET}_\mathsf{Prefix2}$, $\mathsf{NDET}_\mathsf{Rabin}$ and $\mathsf{NDET}_\mathsf{Streett}$, respectively. The union of $\mathsf{NDET}_\mathsf{Prefix1}$ and $\mathsf{NDET}_\mathsf{Prefix2}$ is denoted by $\mathsf{NDET}_\mathsf{Prefix}$.

The corresponding deterministic classes are denoted by DET_G, DET_F, DET_GF, DET_FG, $\mathsf{DET}_\mathsf{Prefix1}$, $\mathsf{DET}_\mathsf{Prefix2}$, $\mathsf{DET}_\mathsf{Prefix}$, $\mathsf{DET}_\mathsf{Rabin}$ and $\mathsf{DET}_\mathsf{Streett}$, respectively. The corresponding total classes are denoted by $\mathsf{NDET}_\mathsf{G}^\mathsf{total}$, $\mathsf{NDET}_\mathsf{F}^\mathsf{total}$ etc. and $\mathsf{DET}_\mathsf{G}^\mathsf{total}$, $\mathsf{DET}_\mathsf{F}^\mathsf{total}$ etc., respectively.

These classes of ω-automata form a hierarchy in terms of expressiveness [35, 36, 49, 58]. For example, there is a $\mathsf{NDET_F}$ automata such that no equivalent $\mathsf{DET_F}$ automata exists. Since $\mathsf{DET_F} \subset \mathsf{NDET_F}$ holds, the class of deterministic liveness automata is strictly less expressive than the class of nondeterministic liveness automata (short $\mathsf{DET_F} \precapprox \mathsf{NDET_F}$). On the other hand, the class of deterministic safety automata is as expressive as the class of nondeterministic safety automata (denoted by $\mathsf{DET_G} \approx \mathsf{NDET_G}$). An important part of the hierarchy of ω-automata can be found in Figure 2.3. Notice that for every class of ω-automata, there is a class of deterministic ω-automata that has the same expressive power.

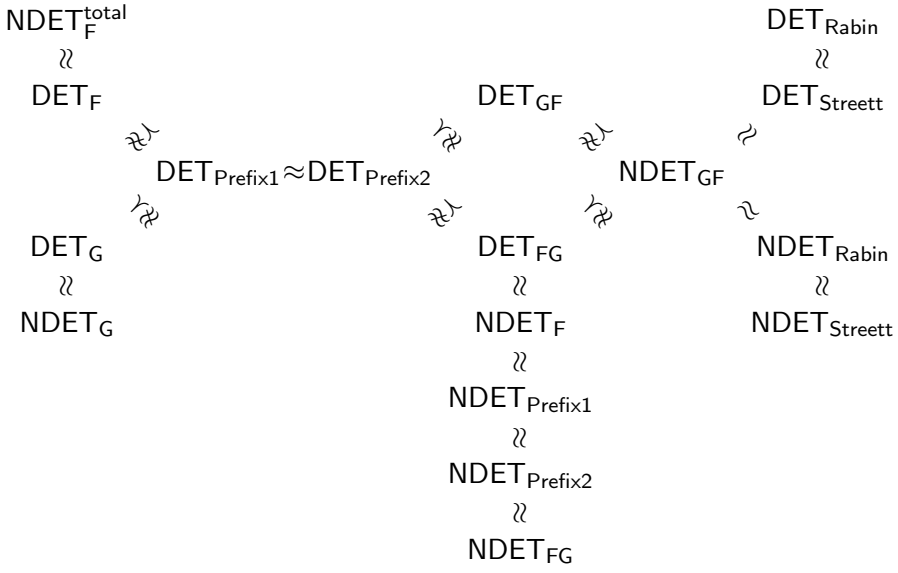

Figure 2.3: Hierarchy of ω-Automata [49]

3 Translation

In the previous chapter, all required formalisms are introduced. Based on this introduction, the translation of PSL to ω-automata can be explained in this chapter. This translation consists of three steps: the translation of PSL to RLTL, the translation of RLTL to LTL and finally, the translation of LTL to ω-automata (Figure 1.1).

3.1 From PSL to RLTL

As mentioned above, the temporal and Boolean layers of PSL consists of FL and OBE. OBE is essentially the well known temporal logic CTL [21]. As it is well known how to handle CTL [19, 49], this work only considers FL.

FL with SEREs is strictly more expressive than LTL. For example, it is well known that there is no LTL formula expressing that a proposition φ holds at every even point of time [59, 60]. However,

$$v \models_{\mathsf{ufl}} \Big(\big(\varphi; \mathsf{true}\big)[*] \mapsto \underline{\mathsf{X}}\big(\varphi! \wedge \underline{\mathsf{X}}\mathsf{true}!\big)\Big) \ \wedge \ \varphi! \wedge \underline{\mathsf{X}}\mathsf{true}!$$

holds for $v \in \mathcal{P}(\mathcal{V})^\omega$ and $\varphi \in \mathsf{prop}_\mathcal{V}$ iff $v^i \models_{\mathsf{prop}} \varphi$ holds[1] for all even i. Since RLTL is as expressive as LTL [5], FL with SEREs cannot be translated to RLTL. Therefore, only SERE-free FL formulas are considered. Moreover, clock-statements are omitted for reasons of simplicity here, since they can be regarded as syntactic sugar as described in Section 2.3. Thus, only the translation of unclocked SERE-free FL (short SUFL) to RLTL is considered.

The semantics of SUFL is quite similar to the semantics of RLTL. There are only two important differences: SUFL is able to consider finite paths, and SUFL uses the special states $\top$ and $\bot$, while RLTL uses acceptance and rejection conditions. The first difference is not important in the scope of this work, because the overall goal is to translate SUFL to ω-automata. Therefore, only infinite paths are of interest. To handle the second difference, the special states $\top$ and $\bot$ are simulated with the acceptance / rejection conditions of RLTL. However, the special states and acceptance / rejection conditions have slightly different semantics. The occurrence of $\top$ or $\bot$ determines whether an arbitrary proposition is fulfilled by the current state. However, the following states are still important. In contrast, if either the acceptance or the rejection condition occurs, the following states are not important according to Lemma 2.2.4. An example showing this difference is $\bot\{p\}^\omega \models_{\mathsf{ufl}} \underline{\mathsf{X}}p$, but $\langle\{r\}\{p\}^\omega, a, r\rangle \not\models^0_{\mathsf{rltl}} \mathsf{X}p$ for $a, r, p \in \mathcal{V}$. To overcome this slightly different semantics only special inputs are considered:

[1] theorem `PSL_WITH_SERES__STRICTLY_MORE_EXPRESSIVE_THAN__LTL__EXAMPLE` in theory `PSLToRLTL`

Definition 3.1.1 (PSL-paths)
A finite PSL-path *over a set of variables $\mathcal{V}$ is a finite word $v \in \mathcal{P}(\mathcal{V})^*$, i. e. a finite word not containing special states. An* infinite PSL-path *over $\mathcal{V}$ is an infinite word $v \in \mathcal{XP}(\mathcal{V})^\omega$ with the following properties:*

- $\forall j.\ v^j{=}\top \longrightarrow v^{j+1}{=}\top$
- $\forall j.\ v^j{=}\bot \longrightarrow v^{j+1}{=}\bot$

The set of all infinite PSL*-paths over $\mathcal{V}$ is denoted by $\mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$. Notice that $\mathcal{P}(\mathcal{V})^\omega \subset \mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$ holds.*

In this work, only infinite PSL-paths are considered. At the first glance, this may seem to be a restriction, however, this is not the case, because essentially only words not containing any special states are of interest. Special states are just used to explain the semantics. Thereby, only paths that fulfil the additional property of PSL-paths are used. In [31], PSL-paths are called *proper words*.

Since paths containing the special states $\top$ and $\bot$ are allowed as input of SUFL formulas, but these special states are not allowed as input of RLTL formulas, both paths and formulas have to be translated. To translate the paths, two new atomic propositions t and b are chosen, i. e. t and b do neither occur in the path nor in the formula. Every occurrence of $\top$ on the path is replaced by the state $\{t\}$. In the same way, every occurrence of $\bot$ is replaced by $\{b\}$. For the formula itself, only minor changes are required: Essentially, only the PSL operators are exchanged with the corresponding RLTL operators. Additionally, t and b are used as acceptance and rejection conditions, respectively, while evaluating the translated formula on the translated path.

Lemma 3.1.2 *With the definitions of Figure 3.1, the relation*

$$v \models_{\mathsf{ufl}} f \iff \langle \mathit{RemoveTopBottom}(t,b,v), t, b\rangle \models^0_{\mathsf{rltl}} \mathit{PSL\_TO\_RLTL}\ f$$

holds[2] *for all $f \in \mathsf{sufl}_{\mathcal{V}}$, all infinite* PSL*-paths $v \in \mathcal{XP}(\mathcal{V})^{\omega^{\top\bot}}$ and all $t, b \notin \mathcal{V}$.*

As t and b never occur at the same point of time on the translated path, this is equivalent[3] *to*

$$v \models_{\mathsf{ufl}} f \iff \mathit{RemoveTopBottom}(t,b,v) \models_{\mathsf{rltl}} \mathsf{ACCEPT}(\mathsf{REJECT}(\mathit{PSL\_TO\_RLTL}\ f, b), t)$$

The proof of Lemma 3.1.2 is based on a structural induction and requires some lemmata about RLTL. In particular, Lemma 2.2.3 and 2.2.4 are important. To express other

[2] theorem `PSL_TO_RLTL_THM` in theory `PSLToRLTL`
[3] theorem `PSL_TO_RLTL___ELIM_ACCEPT_REJECT_THM` in theory `PSLToRLTL`

$$\mathsf{RemoveTopBottom}(t,b,v)^j := \begin{cases} \{t\} & \textit{if } v^j = \top \\ \{b\} & \textit{if } v^j = \bot \\ v^j & \textit{otherwise} \end{cases}$$

function *PSL_TO_RLTL*(Φ)
  case Φ **of**
    b : **return** b;
    $b!$: **return** b;
    $\neg\varphi$: **return** $\neg$PSL_TO_RLTL(φ);
    $\varphi \wedge \psi$: **return** PSL_TO_RLTL$(\varphi) \wedge$ PSL_TO_RLTL(ψ);
    $\underline{\mathsf{X}}\varphi$: **return** $\mathsf{X}\big($PSL_TO_RLTL$(\varphi)\big)$;
    $\varphi\ \underline{\mathsf{U}}\ \psi$: **return** PSL_TO_RLTL$(\varphi)\ \underline{\mathsf{U}}$ PSL_TO_RLTL(ψ);
    φ ABORT b : **return** ACCEPT(PSL_TO_RLTL$(\varphi), b)$;
  end
end

Figure 3.1: Translation of SUFL to RLTL

important properties in a convenient way, some definitions about the occurrence of propositions on a path are needed:

$$\begin{aligned}
\mathsf{NAND_ON_PATH}(v,a,r) &:= \forall t.\, \neg(v^t \models_{\mathsf{prop}} a \ \wedge\ v^t \models_{\mathsf{prop}} r) \\
\mathsf{IS_ON_PATH}(v,p) &:= \exists t.\, v^t \models_{\mathsf{prop}} p \\
\mathsf{BEFORE_ON_PATH}(v,a,b) &:= \forall t.\, (v^t \models_{\mathsf{prop}} b) \Rightarrow \exists t_0.\, (t_0 \leq t \wedge v^{t_0} \models_{\mathsf{prop}} a) \\
\mathsf{BEFORE_ON_PATH_STRONG}(v,a,b) &:= \forall t.\, (v^t \models_{\mathsf{prop}} b) \Rightarrow \exists t_0.\, (t_0 < t \wedge v^{t_0} \models_{\mathsf{prop}} a)
\end{aligned}$$

$\mathsf{NAND\_ON\_PATH}(v,a,r)$ states that a and r never hold at the same point of time on v. As discussed in Section 2.2 that is an important property of pairs of acceptance / rejection conditions. $\mathsf{IS\_ON\_PATH}(v,p)$ means that there is some point of time, where p holds. Therefore, $\mathsf{IS\_ON\_PATH}(v,p)$ holds iff $v \models_{\mathsf{ltl}} \mathsf{F}p$ holds. The expression $\mathsf{BEFORE\_ON\_PATH}(v,a,b)$ means that either b never holds or that a occurs before or at least at the same point of time as b. Finally, the predicate BEFORE_ON_PATH_STRONG is a strong variant of BEFORE_ON_PATH. It demands that a holds strictly before b. Using these definitions, the following lemmata can be easily formulated:

Lemma 3.1.3 *For all $v \in \mathcal{P}(\mathcal{V})^\omega$, $a_1, a_2, r \in \mathsf{prop}_\mathcal{V}$, all $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following holds*[4]*:*

$$\Big(\mathit{NAND\_ON\_PATH}(v^{t\cdot\cdot}, a_1, r) \ \wedge\ \mathit{BEFORE\_ON\_PATH}(v^{t\cdot\cdot}, a_1, a_2)\Big) \Longrightarrow \Big(\langle v, a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \Rightarrow \langle v, a_1, r\rangle \models^t_{\mathsf{rltl}} \varphi\Big)$$

[4] theorem `RLTL_SEM_TIME__ACCEPT_BEFORE_ON_PATH` in theory `ResetLTL_Lemmata`

Informally, this lemma states that valid RLTL formulas do not become invalid if the acceptance condition is strengthened. Since for all words $v \in \mathcal{P}(\mathcal{V})^\omega$, and all propositions $a, b \in \mathsf{prop}_\mathcal{V}$ the expression $\neg\mathsf{BEFORE\_ON\_PATH}(v, a, b)$ is equivalent to $\mathsf{BEFORE\_ON\_PATH\_STRONG}(v, b, a) \wedge \mathsf{IS\_ON\_PATH}(v, b)$, the following lemma is equivalent to Lemma 3.1.3:

Lemma 3.1.4 *For all $v \in \mathcal{P}(\mathcal{V})^\omega$, $a_1, a_2, r \in \mathsf{prop}_\mathcal{V}$, all $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following property holds[5]:*

$$\Big(\mathit{NAND\_ON\_PATH}(v^{t..}, a_1, r) \wedge \langle v, a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \wedge \langle v, a_1, r\rangle \not\models^t_{\mathsf{rltl}} \varphi\Big) \Longrightarrow \Big(\mathit{BEFORE\_ON\_PATH\_STRONG}(v^{t..}, a_2, a_1) \wedge \mathit{IS\_ON\_PATH}(v^{t..}, a_2)\Big)$$

Another important consequence of Lemma 3.1.3 is:

Lemma 3.1.5 *For all $v \in \mathcal{P}(\mathcal{V})^\omega$, $a_1, a_2, r \in \mathsf{prop}_\mathcal{V}$, all $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$, the following property holds[6]:*

$$\Big(\mathit{NAND\_ON\_PATH}(v^{t..}, a_1, r) \wedge \mathit{NAND\_ON\_PATH}(v^{t..}, a_1, r)\Big) \Longrightarrow \Big(\langle v, a_1 \vee a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi \Longleftrightarrow \big(\langle v, a_1, r\rangle \models^t_{\mathsf{rltl}} \varphi \vee \langle v, a_2, r\rangle \models^t_{\mathsf{rltl}} \varphi\big)\Big)$$

Using these lemmata about the acceptance / rejection conditions of RLTL, the proof of Lemma 3.1.2 by structural induction is mainly technical. The cases for $b, b!, \neg\varphi$ and $\varphi \wedge \psi$ are obvious. The case for $\underline{\mathsf{X}}\varphi$ uses the fact that only infinite PSL-paths are considered. The rest it is technical, also the case for $\varphi\ \underline{\mathsf{U}}\ \psi$ is mainly technical. Therefore, the only interesting case is that for φ ABORT b. To prove this case, the presented lemmata about RLTL and a case-analysis are used.

The usage of HOL to formally prove Lemma 3.1.2 has been shown valuable. The case analysis used to prove the case for ABORT is quite tricky. During this case analysis, a small, until then unknown bug in Mike Gordon's deep-embedding of PSL has been discovered: The unclocked semantic of ABORT is defined by $v \models_{\mathsf{ufl}} \varphi$ ABORT b iff either $v \models_{\mathsf{ufl}} \varphi$ or $\exists j.j < |v|$ s.t. $v^j \models_{\mathsf{ufl}} b$ and $v^{0..j-1}\top^\omega \models_{\mathsf{ufl}} \varphi$ holds. This has been literally implemented in HOL. In case of $j = 0$, the word $v^{0..0-1}\top^\omega$ is evaluated to $\top^\omega$ by the formal semantics of PSL. However, because the datatype used to model j in HOL represents natural numbers, $v^{0..0-1}\top^\omega$ evaluated to $v^{0..0}\top^\omega$ and therefore, to $v^0\top^\omega$ in the HOL representation. After reporting this bug to Mike Gordon, it has been fixed.

[5] theorems `RLTL_SEM_TIME__ACCEPT_REJECT_BEFORE_ON_PATH_STRONG` and `RLTL_SEM_TIME__ACCEPT_REJECT_IS_ON_PATH` in theory `ResetLTL_Lemmata`

[6] theorem `RLTL_SEM_TIME___ACCEPT_OR_THM` in theory `ResetLTL_Lemmata`

Lemma 3.1.2 is the central result for the translation of PSL to RLTL. It considers arbitrary infinite PSL-paths as inputs. However, one is usually interested only in paths without special states. Restricting the allowed input paths, Lemma 3.1.2 directly leads to the following theorem:

Theorem 3.1.6 (Translation of SERE-free FL to RLTL)
For all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$ and all $\varphi \in \mathsf{sufl}_\mathcal{V}$[7], the following holds:

$$v \models_{\mathsf{ufl}} \varphi \iff v \models_{\mathsf{rltl}} \mathit{PSL\_TO\_RLTL}(\varphi)$$

Furthermore, PSL_TO_RLTL combined with the rewrite relation $\mathcal{F}$ from Definition 2.3.7 is able to translate clocked formulas[8]*. For all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$, all SERE-free $\varphi \in \mathsf{fl}_\mathcal{V}$ and all clocks $c \in \mathsf{prop}_\mathcal{V}$, the following holds:*

$$v \models^c_{\mathsf{fl}} \varphi \iff v \models_{\mathsf{rltl}} \mathit{PSL\_TO\_RLTL}\big(\mathcal{F}^c(\varphi)\big)$$

3.2 From RLTL to LTL

An important property of RLTL is, that it is as expressive as LTL. Therefore, the translation of RLTL to LTL that is used here has been given together with the original definition of RLTL by Armoni, Bustan, Kupferman and Vardi in 2003 [5]. The correctness of this translation can be easily proved by structural induction.

Theorem 3.2.1 (Translation of RLTL to LTL)
With the definition of Figure 3.2, the following holds[9] *for all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$, all acceptance / rejection conditions $a, r \in \mathsf{prop}_\mathcal{V}$, all RLTL formulas $\varphi \in \mathsf{rltl}_\mathcal{V}$ and all points of time $t \in \mathbb{N}$:*

$$\langle v, a, r\rangle \models^t_{\mathsf{rltl}} \varphi \iff v \models^t_{\mathsf{ltl}} \mathit{RLTL\_TO\_LTL}(a, r, \varphi)$$

At the initial point of time and with the initial pair of acceptance / rejection conditions, this becomes:

$$v \models_{\mathsf{rltl}} \varphi \iff v \models_{\mathsf{ltl}} \mathit{RLTL\_TO\_LTL}(\mathsf{false}, \mathsf{false}, \varphi)$$

3.3 From LTL to ω-Automata

It is well known how LTL can be translated to equivalent ω-automata [20, 24, 25, 52, 61]. In this work, the translation introduced by Schneider [48, 49] is used, because it considers the ω-automaton hierarchy (Figure 2.3). In fact, there are several translations by Schneider: Here, a basic and an optimised version will be used.

[7] theorem `PSL_TO_RLTL___NO_TOP_BOT_THM` in theory `PSLToRLTL`
[8] theorems `PSL_TO_RLTL___CLOCKED_THM` and `PSL_TO_RLTL___NO_TOP_BOT___CLOCKED_THM` in theory `PSLToRLTL`
[9] theorem `RLTL_TO_LTL_THM` in theory `ResetLTL_Lemmata`

3.3.1 Basic Translation

The idea of the basic translation is to translate a LTL formula φ to an ω-automaton $\mathfrak{A}$ by successively abbreviating subformulas of φ with new propositional variables. The value of these new propositional variables is fixed by additional constraints, such that they correspond to the values of the abbreviated LTL formulas. Thus, the translation finally produces a propositional formula p_φ and a set of constraints $\mathcal{C}$ that specify the behaviour of the variables used by p_φ. The set of constraints $\mathcal{C}$ can be expressed as an automaton formula of the form $A_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F})$ such that for all infinite inputs v, there is exactly one run β through this automaton that fulfils $\Phi_\mathcal{F}$, i. e. there is exactly one β with

- $\left(\beta^0 \cup v^0\right) \models_{\mathsf{prop}} \Phi_\mathcal{I}$
- $\left(\beta^i \cup v^i \cup \{\mathtt{X}q \mid q \in \beta^{i+1} \cup v^{i+1}\}\right) \models_{\mathsf{prop}} \Phi_\mathcal{R}$ for all $i \in \mathbb{N}$
- $\beta \cup v \models_{\mathsf{omega}} \Phi_\mathcal{F}$.

This run β is related to the LTL formula φ and propositional formula p_φ by the following relation: $\forall t.\, v \models^t_{\mathsf{ltl}} \varphi \iff (\beta^t \cup v^t) \models_{\mathsf{prop}} p_\varphi$. Therefore, the automaton formula $\mathfrak{A} := A_\exists(Q, \Phi_\mathcal{I}, \Phi_\mathcal{R}, \Phi_\mathcal{F} \wedge p_\varphi)$ is a translation of φ, i. e. for all words v, the relation $v \models_{\mathsf{ltl}} \varphi$ holds iff $v \models_{\mathsf{omega}} \mathfrak{A}$ holds. Furthermore, $\mathfrak{A}$ is equivalent to $A_\exists(Q, \Phi_\mathcal{I} \wedge p_\varphi, \Phi_\mathcal{R}, \Phi_\mathcal{F})$, because p_φ is propositional.

This idea leads to the algorithm described in Figure 3.3. In this work, it is not motivated, how to find suitable constraints. A detailed description and motivation of the algorithm can be found in [49].

Theorem 3.3.1 (Basic translation of LTL to ω-automata)
For all LTL *formulas* $\Phi \in \mathsf{ltl}_\mathcal{V}$ *and* $\hat{A}(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}, p) := \mathit{Streett}(\Phi)$, *where* Streett *is defined as in Figure 3.3, the following holds*[10]*:*

- $p \in \mathsf{prop}_{\mathcal{V} \cup Q}$
- *each* $\xi \in \mathcal{F}$ *is of the form* $\mathsf{GF}\xi'$ *with* $\xi' \in \mathsf{prop}_{\mathcal{V} \cup Q}$
- *for all* $v \in \mathcal{P}(\mathcal{V})^\omega$, *the following holds:*

$$v \models_{\mathsf{ltl}} \Phi \iff v \models_{\mathsf{omega}} \hat{A}(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}, p)$$

- *for all* $v \in \mathcal{P}(\mathcal{V})^\omega$, *the following holds:*

$$v \models_{\mathsf{ltl}} \Phi \iff v \models_{\mathsf{omega}} A_\exists(Q, \mathcal{I} \wedge p, \mathcal{R}, \bigwedge_{\xi \in \mathcal{F}} \xi)$$

[10] several theorems in theory `LTLToOmega`

function *RLTL_TO_LTL*(a, r, Φ)
  case Φ **of**
    b : **return** $a \vee (b \wedge \neg r)$;
    $\neg\varphi$: **return** $\neg\mathsf{RLTL\_TO\_LTL}(r, a, \varphi)$;
    $\varphi \wedge \psi$: **return** $\mathsf{RLTL\_TO\_LTL}(a, r, \varphi) \wedge \mathsf{RLTL\_TO\_LTL}(a, r, \psi)$;
    $\mathsf{X}\varphi$: **return** $a \vee \Big(\mathsf{X}\big(\mathsf{RLTL\_TO\_LTL}(a, r, \varphi)\big) \wedge \neg r\Big)$;
    $\varphi \,\underline{\mathsf{U}}\, \psi$: **return** $\mathsf{RLTL\_TO\_LTL}(a, r, \varphi) \,\underline{\mathsf{U}}\, \mathsf{RLTL\_TO\_LTL}(a, r, \psi)$;
    $\mathsf{ACCEPT}(\varphi, b)$: **return** $\mathsf{RLTL\_TO\_LTL}(a \vee (b \wedge \neg r), r, \varphi)$;
  end
end

Figure 3.2: Translation of RLTL to LTL

function *Streett*(Φ)
  case Φ **of**
    p : **return** $\hat{A}_\exists(\{\}, \mathsf{true}, \mathsf{true}, \{\}, p)$;
    $\neg\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{Streett}(\varphi)$;
      return $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, \neg p_\varphi)$;
    $\varphi \wedge \psi$: **return** $\mathsf{Streett}(\varphi) \times \mathsf{Streett}(\psi)$;
    $\mathsf{X}\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{Streett}(\varphi)$;
      $q := \mathsf{new\_var}$;
      return $\hat{A}_\exists(Q_\varphi \cup \{q\}, \mathcal{I}_\varphi, \mathcal{R}_\varphi \wedge (q \leftrightarrow \mathbf{X}p_\varphi), \mathcal{F}_\varphi, q)$;
    $\varphi \,\underline{\mathsf{U}}\, \psi$: $\hat{A}_\exists(Q_\Phi, \mathcal{I}_\Phi, \mathcal{R}_\Phi, \mathcal{F}_\Phi, p_\varphi \wedge p_\psi) := \mathsf{Streett}(\varphi) \times \mathsf{Streett}(\psi)$;
      $q := \mathsf{new\_var}$;
      $\mathcal{R}_Q := q \leftrightarrow (p_\psi \vee (p_\varphi \wedge \mathtt{X}q))$;
      $\mathcal{F}_Q := \{\mathsf{GF}(q \vee p_\psi)\}$;
      return $\hat{A}_\exists(Q_\Phi \cup \{q\}, \mathcal{I}_\Phi, \mathcal{R}_\Phi \wedge \mathcal{R}_Q, \mathcal{F}_\varphi \cup \mathcal{F}_Q, q)$;
    $\underline{\overleftarrow{\mathsf{X}}}\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{Streett}(\varphi)$;
      $q := \mathsf{new\_var}$;
      return $\hat{A}_\exists(Q_\varphi \cup \{q\}, \mathcal{I}_\varphi \wedge \neg q, \mathcal{R}_\varphi \wedge (\mathtt{X}q \leftrightarrow p_\varphi), \mathcal{F}_\varphi, q)$;
    $\varphi \,\underline{\overleftarrow{\mathsf{U}}}\, \psi$: $\hat{A}_\exists(Q_\Phi, \mathcal{I}_\Phi, \mathcal{R}_\Phi, \mathcal{F}_\Phi, p_\varphi \wedge p_\psi) := \mathsf{Streett}(\varphi) \times \mathsf{Streett}(\psi)$;
      $q := \mathsf{new\_var}$;
      $\mathcal{R}_Q := \mathtt{X}q \leftrightarrow (p_\psi \vee (p_\varphi \wedge q))$;
      $\mathcal{I}_Q := \neg q$;
      return $\hat{A}_\exists(Q_\Phi \cup \{q\}, \mathcal{I}_\Phi \wedge \mathcal{I}_Q, \mathcal{R}_\Phi \wedge \mathcal{R}_Q, \mathcal{F}_\Phi, q)$;
  end
end

Figure 3.3: The basic translation from LTL to ω-automata [49]

The basic translation is able to translate any LTL formula to a $\mathsf{NDET}_{\mathsf{Streett}}$ automaton formula in linear time. That is quite a good result, because in general, this most expressive class of ω-automata is needed. However, for every occurrence of a $\underline{\mathsf{U}}$ operator, a constraint of the form $\mathsf{GF}\varphi$ is added. This is not necessary. In some cases, the constraint can be omitted, in other cases, it can be replaced by a simpler liveness constraint. These improved translations are important to identify a hierarchy of LTL similar to the hierarchy of ω-automata.

In the following section, an improvement is presented that utilises the monotonicity laws of LTL. It is able to omit some constraints. Furthermore, this improvement suffices to identify classes of LTL corresponding to liveness and safety properties. However, due to lack of time, further improvements are not considered in this work.

3.3.2 Improved Translation

Consider the case $\varphi \,\underline{\mathsf{U}}\, \psi$ of the translation described in Figure 3.3. For every input v, there is exactly one run β through the automaton formula $\mathfrak{A} := \mathsf{Streett}(\varphi \,\underline{\mathsf{U}}\, \psi)$. On this run β, the new variable q behaves like the LTL formula $\varphi \,\underline{\mathsf{U}}\, \psi$, i. e. $\forall t.\, \beta^t \models_{\mathsf{prop}} q \Leftrightarrow \beta \models^t_{\mathsf{ltl}} \varphi \,\underline{\mathsf{U}}\, \psi$ holds. If the additional constraint $\mathsf{GF}(q \vee p_\psi)$ is omitted, then there are two possible runs through the resulting automaton[11]: on one run q behaves like $\varphi \,\underline{\mathsf{U}}\, \psi$, on the other run q behaves like $\varphi \,\mathsf{U}\, \psi$. Let $\Phi\langle\varphi \,\underline{\mathsf{U}}\, \psi\rangle_x$ be an arbitrary LTL formula that contains the subformula $\varphi \,\underline{\mathsf{U}}\, \psi$ at a position x. Furthermore, let $\Phi\langle\varphi \,\mathsf{U}\, \psi\rangle_x$ be the formula resulting from this formula by replacing $\varphi \,\underline{\mathsf{U}}\, \psi$ with $\varphi \,\mathsf{U}\, \psi$ at position x. If $\Phi\langle\varphi \,\underline{\mathsf{U}}\, \psi\rangle_x \vee \Phi\langle\varphi \,\mathsf{U}\, \psi\rangle_x$ is equivalent to $\Phi\langle\varphi \,\underline{\mathsf{U}}\, \psi\rangle_x$, the constraint may be omitted when translating the occurrence of $\underline{\mathsf{U}}$ at position x.

For all v and all t, the property $v \models^t_{\mathsf{ltl}} \varphi \,\underline{\mathsf{U}}\, \psi \Longrightarrow v \models^t_{\mathsf{ltl}} \varphi \,\mathsf{U}\, \psi$ holds. All LTL operators except $\neg$ are monotone and $\neg$ is antimonotone. For example, $v \models^t_{\mathsf{ltl}} \neg(\varphi \,\mathsf{U}\, \psi) \Longrightarrow v \models^t_{\mathsf{ltl}} \neg(\varphi \,\underline{\mathsf{U}}\, \psi)$ holds. Therefore, $\neg(\varphi \,\underline{\mathsf{U}}\, \psi) \vee \neg(\varphi \,\mathsf{U}\, \psi)$ is equivalent to $\neg(\varphi \,\underline{\mathsf{U}}\, \psi)$. Exploiting this monotonicity, the constraint $\mathsf{GF}(q \vee p_\psi)$ can be omitted while translating $\varphi \,\underline{\mathsf{U}}\, \psi$, if this formula occurs under an odd number of negations in a larger formula. This leads to the algorithm described in Figure 3.4.

Theorem 3.3.2 (Improved translation of LTL to ω-automata)
For all LTL *formulas* $\Phi \in \mathsf{LTL}_{\mathcal{V}}$ *and* $\hat{A}(Q, \mathcal{I}, \mathcal{R}, \mathcal{F}, p) := \mathit{TopProp}(\Phi)$*, where* *TopProp* *is defined as in Figure 3.4, the following holds*[12]*:*

- $p \in \mathsf{prop}_{\mathcal{V} \cup Q}$
- *each* $\xi \in \mathcal{F}$ *is of the form* $\mathsf{GF}\xi'$ *with* $\xi' \in \mathsf{prop}_{\mathcal{V} \cup Q}$
- *for* $\sigma =$ true *and all* $v \in \mathcal{P}(\mathcal{V})^\omega$*, the following holds:*

$$v \models_{\mathsf{ltl}} \Phi \Longleftrightarrow v \models_{\mathsf{omega}} A_\exists(Q, \mathcal{I} \wedge p, \mathcal{R}, \bigwedge_{\xi \in \mathcal{F}} \xi)$$

[11] theorem `LEMMA_5_35__4` in theory `LTL_LEMMATA`
[12] several theorems in theory `LTLToOmegaOpt`

- *for* $\sigma = \mathsf{false}$ *and all* $v \in \mathcal{P}(\mathcal{V})^\omega$*, the following holds:*

$$v \models_{\mathsf{ltl}} \neg\Phi \Longleftrightarrow v \models_{\mathsf{omega}} A_\exists(Q, \mathcal{I} \wedge \neg p, \mathcal{R}, \bigwedge_{\xi\in\mathcal{F}} \xi)$$

Like the basic translation, the improved translation is able to translate any LTL formula to a $\mathsf{NDET}_{\mathsf{Streett}}$ automaton formula in linear time. In contrast to the basic translation, the acceptance condition may however be simpler. For a certain class of formulas, even no fairness constraints are generated. This will be important when considering the hierarchy of LTL.

function $\mathit{TopProp}_\sigma(\Phi)$
  case Φ **of**
    p : **return** $\hat{A}_\exists(\{\}, \mathsf{true}, \mathsf{true}, \{\}, p)$;
    $\neg\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{TopProp}_{\neg\sigma}(\varphi)$;
      return $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, \neg p_\varphi)$;
    $\varphi \wedge \psi$: **return** $\mathsf{TopProp}_\sigma(\varphi) \times \mathsf{TopProp}_\sigma(\psi)$;
    $\mathsf{X}\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{TopProp}_\sigma(\varphi)$;
      $q := \mathsf{new\_var}$;
      return $\hat{A}_\exists(Q_\varphi \cup \{q\}, \mathcal{I}_\varphi, \mathcal{R}_\varphi \wedge (q \leftrightarrow \mathbf{X} p_\varphi), \mathcal{F}_\varphi, q)$;
    $\varphi \,\underline{\mathsf{U}}\, \psi$: $\hat{A}_\exists(Q_\Phi, \mathcal{I}_\Phi, \mathcal{R}_\Phi, \mathcal{F}_\Phi, p_\varphi \wedge p_\psi) := \mathsf{TopProp}_\sigma(\varphi) \times \mathsf{TopProp}_\sigma(\psi)$;
      $q := \mathsf{new\_var}$;
      $\mathcal{R}_Q := q \leftrightarrow (p_\psi \vee (p_\varphi \wedge \mathtt{X} q))$;
      $\mathcal{F}_Q :=$ **if** σ **then** $\{\mathsf{GF}(q \vee p_\psi)\}$ **else** $\{\}$;
      return $\hat{A}_\exists(Q_\Phi \cup \{q\}, \mathcal{I}_\Phi, \mathcal{R}_\Phi \wedge \mathcal{R}_Q, \mathcal{F}_\varphi \cup \mathcal{F}_Q, q)$;
    $\overleftarrow{\underline{\mathsf{X}}}\varphi$: $\hat{A}_\exists(Q_\varphi, \mathcal{I}_\varphi, \mathcal{R}_\varphi, \mathcal{F}_\varphi, p_\varphi) := \mathsf{TopProp}_\sigma(\varphi)$;
      $q := \mathsf{new\_var}$;
      return $\hat{A}_\exists(Q_\varphi \cup \{q\}, \mathcal{I}_\varphi \wedge \neg q, \mathcal{R}_\varphi \wedge (\mathtt{X} q \leftrightarrow p_\varphi), \mathcal{F}_\varphi, q)$;
    $\varphi \,\overleftarrow{\underline{\mathsf{U}}}\, \psi$: $\hat{A}_\exists(Q_\Phi, \mathcal{I}_\Phi, \mathcal{R}_\Phi, \mathcal{F}_\Phi, p_\varphi \wedge p_\psi) := \mathsf{TopProp}_\sigma(\varphi) \times \mathsf{TopProp}_\sigma(\psi)$;
      $q := \mathsf{new\_var}$;
      $\mathcal{R}_Q := \mathtt{X} q \leftrightarrow (p_\psi \vee (p_\varphi \wedge q))$;
      $\mathcal{I}_Q := \neg q$;
      return $\hat{A}_\exists(Q_\Phi \cup \{q\}, \mathcal{I}_\Phi \wedge \mathcal{I}_Q, \mathcal{R}_\Phi \wedge \mathcal{R}_Q, \mathcal{F}_\Phi, q)$;
  end
end

Figure 3.4: Improved translation of LTL to ω-automata [48, 49]

3.4 Overall Translation

In this chapter, the translation of SERE-free FL formulas to RLTL, the translation of RLTL to LTL and two translations of LTL to automaton formulas have been presented.

The combination of these translations leads to a translation of SERE-free FL formulas to ω-automata.

The translation of SERE-free FL formulas to RLTL and the translation of LTL to automaton formulas require only linear time with respect to the size of the original formula. In contrast, the elimination of the ACCEPT operator, i. e. the translation of RLTL to LTL generates LTL formulas that are quadratic in the size of the original RLTL formulas. For example, consider the following type of RLTL formulas:

$$\mathsf{ACCEPT}(\underbrace{\mathsf{XXX}\ldots}_{n}\varphi, b)$$

The translation to LTL introduces b for every occurrence of the X operator, i. e. n times. Thus, the translation of RLTL to LTL in general needs quadratic time. Due to the translation of RLTL to LTL, the overall translation needs quadratic time with respect to the size of the original formula.

However, these estimations assume that the size of the representation of a LTL formula is linear in the size of the LTL formula. The quadratic blowup of the translation of RLTL to LTL is caused by inserting the acceptance condition b over and over again. An implementation does not need to create multiple copies of b. Structure sharing is able to reduce the required costs to linear time. The translations of LTL formulas to ω-automata can handle all occurrences of an arbitrary subformula at once. Therefore, these translations benefit from structure sharing, too.

The translation of SERE-free FL formulas to RLTL originates to this work. The other parts of the overall translation have already been known [5, 48, 49]. Especially, the translation of LTL to ω-automata is well-investigated. Due to reasons of simplicity, this work only considers a quite simple translation of LTL to ω-automata. There are far more optimised translations [48, 49]. Since the translation of LTL to ω-automata is independent from the other parts of the overall translation, these optimised translations can be easily used in practice when translating SERE-free FL formulas to ω-automata.

4 Temporal Logic Hierarchy for PSL

In Chapter 3, a translation of SERE-free FL to $\mathsf{NDET}_{\mathsf{Streett}}$ automaton formulas is presented. In general, $\mathsf{NDET}_{\mathsf{Streett}}$ automata, the most expressive class of ω-automata is really needed. However, it is interesting to consider subsets of SERE-free FL that can be translated to strictly less expressive classes of ω-automata. Especially, classes of SERE-free FL are interesting that correspond to the classes $\mathsf{DET}_{\mathsf{G}}, \mathsf{DET}_{\mathsf{F}}, \mathsf{DET}_{\mathsf{Prefix}}$ and $\mathsf{DET}_{\mathsf{FG}}$, because the model checking problem of these classes of ω-automata can be solved very efficiently in practice. Additionally, $\mathsf{DET}_{\mathsf{G}}$ and $\mathsf{DET}_{\mathsf{F}}$ are interesting for many applications like simulation or bounded model checking. As motivated in the introduction, this work is therefore especially interested in deterministic liveness and safety automata, i. e. in $\mathsf{DET}_{\mathsf{F}}$ and $\mathsf{DET}_{\mathsf{G}}$ automata.

Similar to the translation of SERE-free FL to RLTL-automata, only unclocked, SERE-free FL (SUFL) is considered for reasons of simplicity. Clocked SERE-free FL formulas can be easily reduced to unclocked ones by the rewrite rules given in Definition 2.3.7. To identify classes of SUFL that correspond to classes of ω-automata, corresponding classes of LTL are considered, first. Then, these classes of LTL are lifted to RLTL and further to SUFL.

4.1 A Hierarchy of LTL

In [49], Schneider identifies classes of LTL that form a hierarchy similar to the hierarchy of ω-automata (Figure 2.3). These classes are defined syntactically according to Figure 4.1. Some elementary, but nevertheless important properties directly follow from this definition:

Lemma 4.1.1 *The following properties hold for the classes of* LTL *defined in Figure 4.1*[1]*:*

- $\Phi \in \mathsf{LTL}_{\mathsf{F}}$ *iff* $\neg\Phi \in \mathsf{LTL}_{\mathsf{G}}$ *and* $\Phi \in \mathsf{LTL}_{\mathsf{G}}$ *iff* $\neg\Phi \in \mathsf{LTL}_{\mathsf{F}}$
- $\Phi \in \mathsf{LTL}_{\mathsf{FG}}$ *iff* $\neg\Phi \in \mathsf{LTL}_{\mathsf{GF}}$ *and* $\Phi \in \mathsf{LTL}_{\mathsf{GF}}$ *iff* $\neg\Phi \in \mathsf{LTL}_{\mathsf{FG}}$
- $\mathsf{LTL}_{\mathsf{F}} \subseteq \mathsf{LTL}_{\mathsf{GF}} \cap \mathsf{LTL}_{\mathsf{FG}}$
- $\mathsf{LTL}_{\mathsf{G}} \subseteq \mathsf{LTL}_{\mathsf{GF}} \cap \mathsf{LTL}_{\mathsf{FG}}$
- $\mathsf{LTL}_{\mathsf{Prefix}} \subseteq \mathsf{LTL}_{\mathsf{GF}} \cap \mathsf{LTL}_{\mathsf{FG}}$

[1] theorem `IS_LTL_RELATIONS` in theory `LTL`

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{G}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{G}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{G}} & = & \varphi \in \mathsf{LTL}_{\mathsf{G}} \wedge \psi \in \mathsf{LTL}_{\mathsf{G}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{G}} & = & \varphi \in \mathsf{LTL}_{\mathsf{G}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{G}} & = & \mathsf{false} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{G}} & = & \varphi \in \mathsf{LTL}_{\mathsf{G}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{G}} & = & \varphi \in \mathsf{LTL}_{\mathsf{G}} \wedge \psi \in \mathsf{LTL}_{\mathsf{G}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{F}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{G}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \wedge \psi \in \mathsf{LTL}_{\mathsf{F}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \wedge \psi \in \mathsf{LTL}_{\mathsf{F}} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{F}} & = & \varphi \in \mathsf{LTL}_{\mathsf{F}} \wedge \psi \in \mathsf{LTL}_{\mathsf{F}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{GF}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \wedge \psi \in \mathsf{LTL}_{\mathsf{GF}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \wedge \psi \in \mathsf{LTL}_{\mathsf{F}} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \wedge \psi \in \mathsf{LTL}_{\mathsf{GF}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{FG}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{GF}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \wedge \psi \in \mathsf{LTL}_{\mathsf{FG}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \wedge \psi \in \mathsf{LTL}_{\mathsf{FG}} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{LTL}_{\mathsf{FG}} \wedge \psi \in \mathsf{LTL}_{\mathsf{FG}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{Prefix}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{LTL}_{\mathsf{Prefix}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{LTL}_{\mathsf{Prefix}} \wedge \psi \in \mathsf{LTL}_{\mathsf{Prefix}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{G}} \cup \mathsf{LTL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{G}} \cup \mathsf{LTL}_{\mathsf{F}} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{G}} \cup \mathsf{LTL}_{\mathsf{F}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{Prefix}} & = & \varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{G}} \cup \mathsf{LTL}_{\mathsf{F}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{LTL}_{\mathsf{Streett}} & & \\
\neg\varphi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{LTL}_{\mathsf{Streett}} \\
\varphi \wedge \psi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{LTL}_{\mathsf{Streett}} \wedge \psi \in \mathsf{LTL}_{\mathsf{Streett}} \\
\mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \mathsf{X}\varphi \in \mathsf{LTL}_{\mathsf{GF}} \cup \mathsf{LTL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{LTL}_{\mathsf{GF}} \cup \mathsf{LTL}_{\mathsf{FG}} \\
\overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \overline{\underline{\mathsf{X}}}\varphi \in \mathsf{LTL}_{\mathsf{GF}} \cup \mathsf{LTL}_{\mathsf{FG}} \\
\varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{Streett}} & = & \varphi \,\overline{\underline{\mathsf{U}}}\, \psi \in \mathsf{LTL}_{\mathsf{GF}} \cup \mathsf{LTL}_{\mathsf{FG}}
\end{array}
$$

Figure 4.1: Classes of LTL

Additionally, it is easy to see, that $\mathsf{LTL}_{\mathsf{Prefix}}$ is the Boolean closure of $\mathsf{LTL}_{\mathsf{G}}$ and $\mathsf{LTL}_{\mathsf{F}}$ and that $\mathsf{LTL}_{\mathsf{Streett}}$ is the Boolean closure of $\mathsf{LTL}_{\mathsf{GF}}$ and $\mathsf{LTL}_{\mathsf{FG}}$.

These classes of LTL can be translated to the corresponding classes of deterministic ω-automata:

Lemma 4.1.2 *For any* $\Phi \in \mathsf{LTL}_{\kappa}$ *with* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, *there is an equivalent* ω*-automaton* $\mathfrak{A}_{\Phi} \in \mathsf{Det}_{\kappa}$.

Proof The improved translation of LTL to ω-automata presented in Section 3.3.2 is able to translate every $\mathsf{LTL}_{\mathsf{G}}$ formula to an equivalent $\mathsf{NDET}_{\mathsf{G}}$ automaton[2]. Since $\mathsf{DET}_{\mathsf{G}}$ is as expressive as $\mathsf{NDET}_{\mathsf{G}}$ (see Figure 2.3), every $\mathsf{LTL}_{\mathsf{G}}$ formula can be translated to an equivalent $\mathsf{DET}_{\mathsf{G}}$ automaton.

According to Lemma 4.1.1, $\neg\Phi \in \mathsf{LTL}_{\mathsf{G}}$ holds for all $\Phi \in \mathsf{LTL}_{\mathsf{F}}$. Therefore, $\neg\Phi$ can be translated to an equivalent $\mathsf{DET}_{\mathsf{G}}$ automaton. The negation of this $\mathsf{DET}_{\mathsf{G}}$ automaton leads to a $\mathsf{DET}_{\mathsf{F}}$ automaton. Thus, every $\Phi \in \mathsf{LTL}_{\mathsf{F}}$ can be translated to an equivalent $\mathsf{DET}_{\mathsf{F}}$ automaton.

$\mathsf{LTL}_{\mathsf{Prefix}}$ is the Boolean closure of $\mathsf{LTL}_{\mathsf{G}}$ and $\mathsf{LTL}_{\mathsf{F}}$. Analogously, $\mathsf{DET}_{\mathsf{Prefix}}$ is the Boolean closure of $\mathsf{DET}_{\mathsf{G}}$ and $\mathsf{DET}_{\mathsf{F}}$ [49]. Therefore, every $\Phi \in \mathsf{LTL}_{\mathsf{Prefix}}$ can be translated to an equivalent $\mathsf{DET}_{\mathsf{Prefix}}$ automaton by translating all subformulas $\Phi_G \in \mathsf{DET}_{\mathsf{G}}$,

[2] theorem `LTL_TO_OMEGA_OPTIMIZED_THM___IS_LTL_G` in theory `LTLToOmegaOpt`

$\Phi_F \in \mathsf{DET_F}$ and applying the Boolean operations discussed in Section 2.4.6 to the resulting automaton formulas. Also the remaining classes $\mathsf{LTL_{GF}}$, $\mathsf{LTL_{FG}}$ and $\mathsf{LTL_{Streett}}$ can be translated to the corresponding classes of deterministic ω-automata [49]. The proof is omitted here.

ω-automata are strictly more expressive than LTL. In general, automata are able to count modulo a constant number by storing information in their states. This ability to count is the difference between the expressiveness of ω-automata and LTL. LTL is as expressive as *noncounting* ω-automata [40]. As for $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$ each class DET_κ contains counting automata, LTL_κ is strictly less expressive than DET_κ. However, LTL_κ is as expressive as noncounting DET_κ automata (denoted by TDET_κ) [49].

Lemma 4.1.3 *For any $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, the logic LTL_κ is complete with respect to noncounting ω-automata, i. e. LTL_κ is at least as expressive as TDET_κ (denoted by $\mathsf{TDET}_\kappa \precsim \mathsf{LTL}_\kappa$). Together with Lemma 4.1.2, this implies $\mathsf{LTL}_\kappa \approx \mathsf{TDET}_\kappa$.*

Lemma 4.1.4 *The classes of noncounting deterministic ω-automata form a similar hierarchy as the classes of deterministic ω-automata [49]. This hierarchy is shown in Figure 4.2.*

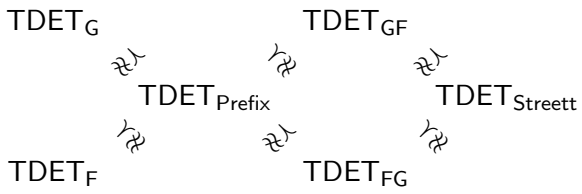

Figure 4.2: Hierarchy of deterministic noncounting ω automata [49]

Therefore, the classes of LTL form a hierarchy as well. Moreover, the same hierarchy is formed by classes of LTL without past operators, i. e. by the classes of FutureLTL. FutureLTL is as expressive as LTL [23], i. e. for each LTL formula φ, there is an FutureLTL formula $\varphi_{\mathsf{future}}$ that is initially equivalent to φ. However, there is in general no FutureLTL formula that is equivalent to φ. This relation between LTL and FutureLTL holds for the defined classes of LTL, as well [49]. For $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, let $\mathsf{FutureLTL}_\kappa$ denote the future fragment of LTL_κ, i. e. $\mathsf{FutureLTL}_\kappa := \mathsf{LTL}_\kappa \cap \mathsf{FutureLTL}$. Using these notations, the following lemma holds:

Lemma 4.1.5 *For any $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, the logic LTL_κ is as expressive as $\mathsf{FutureLTL}_\kappa$ [49]. Therefore, $\mathsf{FutureLTL}_\kappa$ is as expressive as TDET_κ.*

4.2 A Hierarchy of RLTL

The algorithm presented in Section 3.2 translates a RLTL formula to an equivalent LTL formula by introducing additional propositional subformulas. However, no additional temporal operators are introduced. Since propositional subformulas do not affect the membership in a LTL class, it is easy to define classes of RLTL that can be translated to the corresponding LTL class. To prove the completeness of these classes, it remains to show that every LTL class is at most as expressive as the corresponding class of RLTL.

$$
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{G}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{G}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{F}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{G}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{G}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{G}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{G}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{G}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{G}} & = & \mathsf{false} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{G}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{G}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{F}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{F}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{G}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{F}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{F}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{F}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{F}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{F}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{F}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{F}} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{F}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{F}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{GF}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{FG}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{GF}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{GF}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{GF}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{GF}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{F}} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{GF}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{GF}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{FG}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{GF}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{FG}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{FG}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{FG}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{FG}} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{FG}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{FG}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{Prefix}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Prefix}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Prefix}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{Prefix}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{Prefix}} & = & \mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{G}} \cup \mathsf{RLTL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{Prefix}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{G}} \cup \mathsf{RLTL}_{\mathsf{F}} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Prefix}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{RLTL}_{\mathsf{Streett}} & & \\
\neg\varphi \in \mathsf{RLTL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Streett}} \\
\varphi \wedge \psi \in \mathsf{RLTL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Streett}} \wedge \psi \in \mathsf{RLTL}_{\mathsf{Streett}} \\
\mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{Streett}} & = & \mathsf{X}\varphi \in \mathsf{RLTL}_{\mathsf{GF}} \cup \mathsf{RLTL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{Streett}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{RLTL}_{\mathsf{GF}} \cup \mathsf{RLTL}_{\mathsf{FG}} \\
\mathsf{ACCEPT}(\varphi, b) \in \mathsf{RLTL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{RLTL}_{\mathsf{Streett}}
\end{array}
$$

Figure 4.3: Classes of RLTL

Lemma 4.2.1 *Let RLTL_TO_LTL be the function defined in Figure 3.2, and let* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$ *hold. Then, with the definitions of the classes of* RLTL *from Figure 4.3 for all* RLTL *formulas* Φ *and all acceptance / rejection conditions* a, r, *the following holds*[3]*:*

$$\Phi \in \mathsf{RLTL}_{\kappa} \Longleftrightarrow \mathit{RLTL\_TO\_LTL}(a, r, \Phi) \in \mathsf{LTL}_{\kappa}$$

Thus, according to Theorem 3.2.1 for every $\Phi \in \mathsf{RLTL}_{\kappa}$*, there is an equivalent* $\Phi_{\mathsf{ltl}} \in \mathsf{LTL}_{\kappa}$.

This lemma implies $\mathsf{RLTL}_{\kappa} \precapprox \mathsf{LTL}_{\kappa}$ for $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$. In order to prove $\mathsf{RLTL}_{\kappa} \approx \mathsf{LTL}_{\kappa}$, it remains to show, that $\mathsf{RLTL}_{\kappa} \succapprox \mathsf{LTL}_{\kappa}$ holds. According to Lemma 4.1.5, $\mathsf{FutureLTL}_{\kappa}$ is as expressive as LTL_{κ}. Therefore, it is sufficient to show

[3] theorem `IS_RLTL_LTL_THM` in theory `ResetLTL_Lemmata`

$\mathsf{FutureLTL}_\kappa \precnsim \mathsf{RLTL}_\kappa$. This can be shown by translating every $\Phi \in \mathsf{FutureLTL}_\kappa$ to an equivalent $\Phi_{\mathsf{rltl}} \in \mathsf{RLTL}_\kappa$. Since FutureLTL is a subset of RLTL, it is straightforward to find such a translation.

Lemma 4.2.2 *With the definition of Figure 4.4, the following holds for all infinite words $v \in \mathcal{P}(\mathcal{V})^\omega$ and all* FutureLTL *formulas $\varphi \in \mathsf{ltl}_\mathcal{V}$[4]:*

$$v \models_{\mathsf{ltl}} \varphi \iff v \models_{\mathsf{rltl}} \mathit{LTL\_TO\_RLTL}(\varphi)$$

Furthermore, LTL_TO_RLTL translates a class of FutureLTL *to the corresponding class of* RLTL*, i. e. for each $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$ and all* FutureLTL *formulas Φ, the following holds:*

$$\Phi \in \mathsf{FutureLTL}_\kappa \iff \mathit{LTL\_TO\_RLTL}(\varphi) \in \mathsf{RLTL}_\kappa$$

```
function LTL_TO_RLTL(Φ)
    case Φ of
        b          : return b;
        ¬φ         : return ¬LTL_TO_RLTL(φ);
        φ ∧ ψ      : return LTL_TO_RLTL(φ) ∧ LTL_TO_RLTL(ψ);
        X φ        : return X(LTL_TO_RLTL(φ));
        φ U ψ      : return LTL_TO_RLTL(φ) U LTL_TO_RLTL(ψ);
    end
end
```

Figure 4.4: Translation of FutureLTL to RLTL

As motivated, this directly leads to the following theorem:

Theorem 4.2.3 (Hierarchy of RLTL)
For any $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, the logic RLTL_κ is as expressive as LTL_κ. Therefore, RLTL_κ is complete with respect to noncounting ω-automata, i. e. RLTL_κ is as expressive as TDET_κ.

4.3 A Hierarchy of PSL

The translation of SUFL to RLTL discussed in Section 3.1 essentially replaces every PSL operator with its corresponding RLTL operator. Therefore, it is straightforward to identify classes of SUFL that correspond to the classes of RLTL.

[4] theorem `FUTURE_LTL_TO_RLTL_THM` in theory `ResetLTL_Lemmata`

$$
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{G}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{G}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{G}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{G}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{G}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{G}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{G}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{G}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{G}} & = & \mathsf{false} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{G}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{G}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{F}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{F}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{F}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{G}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{F}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{F}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{F}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{F}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{F}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{F}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{F}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{F}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{GF}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{GF}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{GF}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{FG}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{GF}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{GF}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{GF}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{GF}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{GF}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{GF}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{GF}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{GF}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{GF}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{FG}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{FG}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{FG}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{GF}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{FG}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{FG}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{FG}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{FG}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{FG}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{FG}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{FG}} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{FG}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{FG}}
\end{array}
$$

$$
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{Prefix}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{Prefix}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Prefix}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Prefix}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{Prefix}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{Prefix}} & = & \mathsf{X}\varphi \in \mathsf{SUFL}_{\mathsf{G}} \cup \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{Prefix}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{G}} \cup \mathsf{SUFL}_{\mathsf{F}} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{Prefix}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Prefix}}
\end{array}
\qquad
\begin{array}{rcl}
b \in \mathsf{SUFL}_{\mathsf{Streett}} & & \\
b! \in \mathsf{SUFL}_{\mathsf{Streett}} & & \\
\neg\varphi \in \mathsf{SUFL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Streett}} \\
\varphi \wedge \psi \in \mathsf{SUFL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Streett}} \wedge \psi \in \mathsf{SUFL}_{\mathsf{Streett}} \\
\underline{\mathsf{X}}\varphi \in \mathsf{SUFL}_{\mathsf{Streett}} & = & \mathsf{X}\varphi \in \mathsf{SUFL}_{\mathsf{GF}} \cup \mathsf{SUFL}_{\mathsf{FG}} \\
\varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{Streett}} & = & \varphi \,\underline{\mathsf{U}}\, \psi \in \mathsf{SUFL}_{\mathsf{GF}} \cup \mathsf{SUFL}_{\mathsf{FG}} \\
\varphi \text{ ABORT } b \in \mathsf{SUFL}_{\mathsf{Streett}} & = & \varphi \in \mathsf{SUFL}_{\mathsf{Streett}}
\end{array}
$$

Figure 4.5: Classes of SUFL

Lemma 4.3.1 *Let PSL_TO_RLTL be the function defined in Figure 3.1 and let* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$ *hold. Then, with the definitions of the classes of* SUFL *from Figure 4.5 for all* SUFL *formulas* Φ*, the following holds*[5]*:*

$$\Phi \in \mathsf{SUFL}_{\kappa} \iff \mathit{PSL\_TO\_RLTL}(\Phi) \in \mathsf{RLTL}_{\kappa}$$

Thus, according to Theorem 3.1.6 for every $\Phi \in \mathsf{RLTL}_{\kappa}$*, there is a* $\Phi_{\mathsf{rltl}} \in \mathsf{RLTL}_{\kappa}$ *that is equivalent on infinite paths without special states.*

Therefore, for each $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$, the class SUFL_{κ} is at most as expressive as RLTL_{κ}, if only infinite paths without special states are considered. This condition that only infinite paths without special states are considered is always assumed in the following.

It remains to show that SUFL_{κ} is at least as expressive as RLTL_{κ}. Since RLTL_{κ} is as expressive as $\mathsf{FutureLTL}_{\kappa}$, it is sufficient to translate all $\Phi \in \mathsf{FutureLTL}_{\kappa}$ to equivalent $\Phi_{\mathsf{sufl}} \in \mathsf{SUFL}_{\kappa}$. As FutureLTL is a subset of SUFL, it is straightforward to find such a translation:

Lemma 4.3.2 *Let LTL_TO_PSL be the function defined in Figure 4.6. Then, the following holds for all infinite words* $v \in \mathcal{P}(\mathcal{V})^{\omega}$ *and all* FutureLTL *formulas* $\varphi \in \mathsf{ltl}_{\mathcal{V}}$[6]*:*

$$v \models_{\mathsf{ltl}} \varphi \iff v \models_{\mathsf{ufl}} \mathit{LTL\_TO\_PSL}(\varphi)$$

[5] theorem `IS_PSL_RLTL_THM` in theory `PSLToRLTL`
[6] theorem `FUTURE_LTL_TO_PSL_THM` in theory `PSLToRLTL`

Furthermore, LTL_TO_PSL translates all classes of FutureLTL *to the corresponding class of* SUFL*, i. e. for each* $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$ *and all* FutureLTL *formulas* Φ*, the following holds:*

$$\Phi \in \mathsf{FutureLTL}_\kappa \iff \mathit{LTL\_TO\_PSL}(\varphi) \in \mathsf{SUFL}_\kappa$$

function *LTL_TO_PSL*(Φ)
  case Φ **of**
    b : **return** b!;
    $\neg\varphi$: **return** $\neg$LTL_TO_PSL(φ);
    $\varphi \wedge \psi$: **return** LTL_TO_PSL(φ) $\wedge$ LTL_TO_PSL(ψ);
    $\mathsf{X}\varphi$: **return** $\underline{\mathsf{X}}$(LTL_TO_PSL(φ));
    $\varphi\ \underline{\mathsf{U}}\ \psi$: **return** LTL_TO_PSL(φ) $\underline{\mathsf{U}}$ LTL_TO_PSL(ψ);
  end
end

Figure 4.6: Translation of FutureLTL to SUFL

Therefore, SUFL_κ is as expressive as RLTL_κ. The combination with the other results of this chapter leads to the following theorem:

Theorem 4.3.3 (Hierarchy of PSL)
For any $\kappa \in \{\mathsf{G}, \mathsf{F}, \mathsf{Prefix}, \mathsf{GF}, \mathsf{FG}, \mathsf{Streett}\}$*, the logics* LTL_κ*,* $\mathsf{FutureLTL}_\kappa$*,* RLTL_κ *and* SUFL_κ *are as expressive as* TDET_κ*. Furthermore,* LTL*,* FutureLTL*,* RLTL *and* SUFL *are as expressive as* $\mathsf{TDET}_{\mathsf{Streett}}$*. This leads to the hierarchy shown in Figure 4.7.*

This theorem is the main result concerning the hierarchy of PSL. Since efficient translations between the classes of PSL and the classes deterministic noncounting ω-automata are given, it is of immediate practical relevance.

However, Theorem 4.3.3 also shows the limits of the defined hierarchy. The classes of LTL and PSL are as expressive as the corresponding classes of noncounting, deterministic ω-automata. For LTL, that is the best possible result, since LTL itself is not able to count. Therefore, the classes of LTL are in some sense complete with respect to the expressiveness of LTL. For example, every safety property that can be expressed by LTL can also be expressed by LTL_G. This situation is different for PSL. In Section 3.1, a property is presented that can be expressed by unclocked FL with SEREs, but not by LTL. Since LTL is as expressive as unclocked, SERE-free FL, this example shows that unclocked FL with SEREs is strictly more expressive than unclocked, SERE-free FL. Moreover, the property is a safety property that is expressible by unclocked FL with SEREs, but not by SUFL_G. Therefore, even if only infinite paths without special

states are considered, the defined classes of unclocked FL are not able to express all properties that can be expressed as well by unclocked FL as by the corresponding class of deterministic ω-automata.

The hierarchy of PSL only considers unclocked formulas. Clocked SERE-free FL can be rewritten to unclocked SERE-free FL by the rewrite relation $\mathcal{F}$ defined in Definition 2.3.7. However, $\mathcal{F}$ introduces additional U and $\underline{\mathsf{U}}$ operators. These additional operators influence the membership of a formula in the defined classes of PSL. Especially, the operator U is problematic. For two formulas $\varphi, \psi \in \mathsf{SUFL}$, the formula $\varphi \mathsf{U} \psi$ is defined as a shorthand for $\varphi \mathbin{\underline{\mathsf{U}}} \psi \vee \neg(\mathsf{true} \mathbin{\underline{\mathsf{U}}} \varphi)$. For arbitrary φ, ψ, this formula neither belongs to TDET_F nor to TDET_G. However, $\varphi \mathsf{U} \psi$ is on infinite PSL-paths equivalent to $\neg\Big(\neg\psi \mathbin{\underline{\mathsf{U}}} (\neg\varphi \wedge \neg\psi)\Big)$[7] [31]. This formula belongs to SUFL_G iff $\varphi, \psi \in \mathsf{SUFL}_\mathsf{G}$. It can be used as an alternative definition of the semantics of U, if only infinite PSL-paths are considered. However, a better solution is to use U like a basic operator: The translation of PSL to RLTL, the translation of RLTL to LTL and the translations of LTL to ω-automata should be extended by cases for U. Additionally, the definitions of the PSL, RLTL and LTL classes should be extended to consider U. Using such extensions, U can be handled in the best possible way.

[7] theorem `PSL_WEAK_UNTIL__ALTERNATIVE_DEF` in theory `PSLToRLTL`

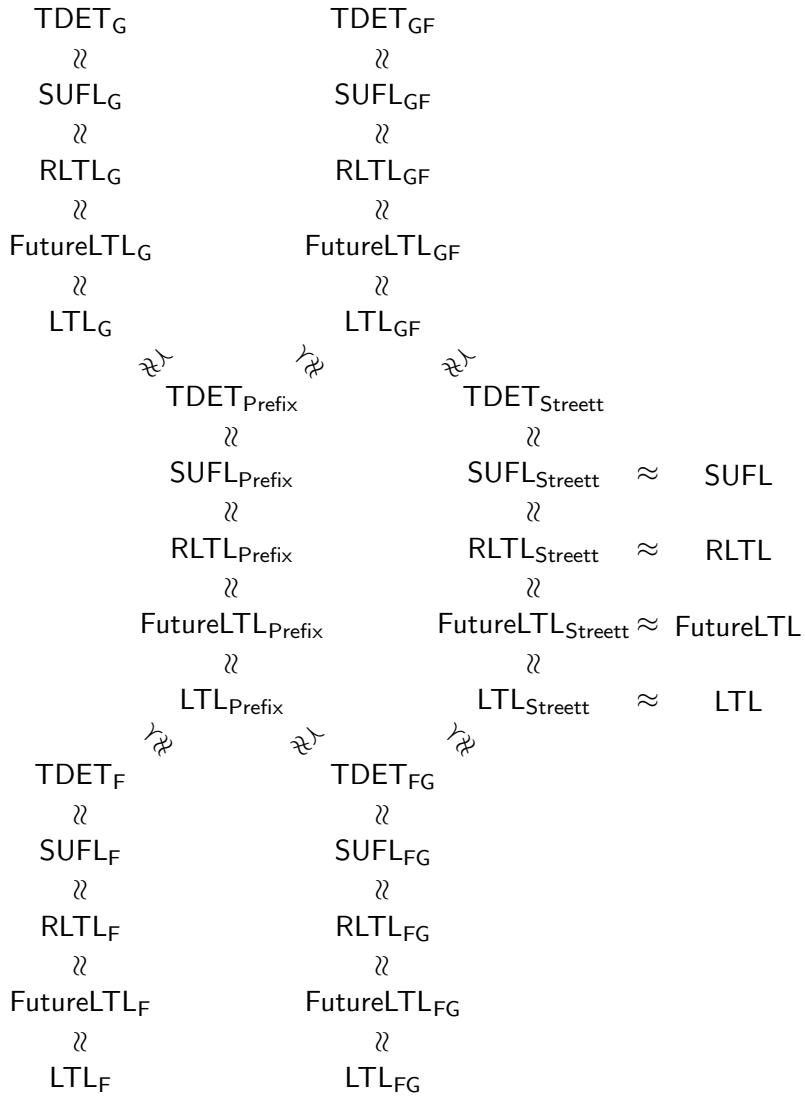

Figure 4.7: Hierarchy of PSL

5 The HOL System

The HOL System [27, 29] is an interactive theorem prover for higher order logic. The version of higher order logic used in HOL is predicate calculus with terms from the typed lambda calculus [17]. The interactive frontend of HOL is formed by the functional programming language ML, in which terms and theorems of the logic, proof strategies and logical theories are implemented.

The first antecedent of HOL is LCF (logic for computable functions) [30]. The original LCF was implemented at Edinburgh in the early 1970's, and is now referred to as *Edinburgh* LCF. Further developments at Cambridge led to a version of LCF called *Cambridge* LCF. The HOL System is implemented on top of an early version of Cambridge LCF, and consequently, many features of both Edinburgh and Cambridge LCF were inherited by HOL. An enhanced and rationalised version, called HOL88 was released in 1988 after the original HOL System had been in use for several years. Other important releases of HOL are HOL90 and HOL98. There is also a commercial version of HOL, called *ProofPower*, which was developed by ICL Secure Systems. In this work, HOL4 is used, which is a successor to the mentioned versions of HOL.

The HOL library created in this work is structured into several theories. The theory `TemporalModel` contains embeddings of basic constructs like words or propositional logic. Furthermore, it contains definitions of important functions and predicates related to these constructs. There are also some elementary lemmata. The theories `LTL`, `ResetLTL` and `Omega_Automata` contain the same for LTL, RLTL and automaton formulas.

The definitions and lemmata of these theories are used by `TemporalLib`. This library defines some simplification sets and specialised tactics. `TemporalLib` also uses the theory `GeneralLemmata`, which contains some simple lemmata for rewriting that are not directly related with the central formalisms. Most of these lemmata are about operations on sets.

`TemporalLib` is used to prove some more complicated lemmata about the basic constructs, LTL, RLTL and automaton formulas. The resulting theories are `TemporalModel_Lemmata`, `LTL_Lemmata`, `ResetLTL_Lemmata` and `Omega_Automata_Lemmata`. The basic translation of LTL to ω-automata can be found in the theory `LTLToOmega`. The correctness of the improved translation is proved in the theory `LTLToOmegaOpt`. Since the translation of RLTL to LTL is comparable simple, it is part of the theory `ResetLTL_Lemmata`. Finally, definitions and lemmata about PSL and the translation of SUFL to RLTL can be found in the theory `PSLToRLTL`.

In the following, the deep embeddings of the used formalisms and some important

theorems are presented. Additionally, the used parts of Mike Gordon's PSL library [26] are discussed. However, these presentations are quite short, because the definitions and theorems used in HOL are in general very similar to the ones presented in the previous chapters.

5.1 Deep Embedding of PSL

Mike Gordon's PSL theory [26, 28] is intended to be as close to the formal semantics of PSL as possible. Let `'prop` be an arbitrary HOL type. Then, the HOL type `'prop letter` is defined to be either one of the special states `TOP`, `BOTTOM` or a state `STATE s`, where `s` is a set over `'prop`. Propositional logic formulas over variables of type `'prop` are formalised by the HOL type `'prop bexp`. The predicate `B_SEM`: `'prop letter` → `'prop bexp` → `bool` defines the semantics of these propositional formulas.

A path over an arbitrary HOL type `'a` is modelled by the HOL type `'a path`. Finite paths are denoted by `FINITE p`, where `p` is a `'a list`. Infinite paths are denoted by `INFINITE p`, where `p` is a function of type `num` → `'a`.

For an arbitrary HOL type `'prop`, the HOL type `'prop fl` is a deep embedding of FL over variables of type `'prop`. The unclocked semantics of `'prop fl` is defined by the predicate `UF_SEM`: `'prop letter path` → `'prop fl` → `bool`.

Another part of the PSL embedding is the theorem `fl_induct`, which is used for structural inductions over `'prop fl` formulas. There are of course a lot of other definitions and theorems. Especially, SEREs have not been mentioned yet at all. However, these are the most important definitions and theorems for this work.

5.2 Deep Embedding of LTL and RLTL

In contrast to PSL, the formalisms LTL and RLTL do not use special states and consider only finite states. Therefore, the definitions of propositional logic and paths can be simplified. For an arbitrary HOL type `'prop`, a state is modelled by the HOL type `'prop temporal_state`. It is defined as a set over `'prop`. Propositional logic over variables of type `'prop` is modelled by the HOL type `'prop prop_logic`. Its semantics is defined by the predicate `P_SEM`. An infinite path over an arbitrary HOL type `'a` is modelled by the HOL type `'a temporal_path`. It is defined as an abbreviation of the HOL type `num` → `'prop temporal_state`.

The syntax and semantics are defined exactly as described in Chapter 2. LTL and RLTL over variables of type `'prop` are modelled by the HOL types `'prop ltl` and `'prop rltl`. Their semantics is assigned by the predicates `LTL_SEM_TIME` and `RLTL_SEM_TIME`. The semantics at the initial point of time is referenced by the predicates `LTL_SEM` and `RLTL_SEM`.

5.3 Deep Embedding of ω-Automata

The deep embedding of ω-automata uses the same definitions of paths and propositional logic as the embeddings of LTL and RLTL. To define automaton formulas, a deep embedding of a symbolic representation of semiautomata is used.

To define the transition relation of semiautomata, which may use the special variables `X`q to refer to the value of the variable q at the next point of time, extended propositional logic is introduced by the HOL type `'prop xprop_logic` and the predicate `XP_SEM`: `'prop temporal_state` × `'prop temporal_state` → `'prop xprop_logic` → `bool`. These extended propositional logic is used to define semiautomata. The HOL type `'prop semi_automaton` is defined as a tuple of a set of state variables, the set of initial states modelled by a `'prop prop_logic` formula and the translation relation modelled by a `'prop xprop_logic` formula. The semantics of semiautomata is modelled by the predicate `RUN` that checks, whether a path is a run of some input through a semiautomaton. Thereby, the extended syntax and semantics of automaton formulas presented in Section 2.4.5 are used, i. e. input variables are handled like state variables. However, the definition of the syntax of automaton formulas demands that the sets of input and state variables are disjoint. This is not guaranteed by the HOL-embedding. Instead of this, state variables occurring as inputs are ignored by the definition of the semantics.

This deep embedding of semiautomata is used to define automaton formulas. Flat acceptance conditions are modelled by the HOL type `automaton_formula`. Automaton formulas are modelled by the HOL type `automaton_formula`. Their semantics are assigned by the predicates `ACCEPT_COND_SEM_TIME`, `ACCEPT_COND_SEM` and `A_SEM`.

In contrast to the formal definition, the syntax of the embedding of automaton formulas in HOL does not guarantee that state and input variables are disjoint. Therefore, the predicate `VARDISJOINT_AUTOMATON_FORMULA` checks whether this condition is met. Furthermore, there are some definitions and theorems that allow variable renamings.

5.4 Translations of LTL to ω-Automata

The translations of LTL to ω-automata are quite tricky. The correctness of both, the basic and the improved translation, is proved by structural induction. Since the proof becomes very large, the cases of this structural induction are proved as separate theorems. Thereby, the induction hypothesis is abstracted by special predicates `BASIC_TRANSLATION_INVARIANTS` and `OPTIMIZED_TRANSLATION_INVARIANTS`.

The proofs of the correctness of the cases is quite tricky. A lot of lemmata about ω-automata and LTL are used. Moreover, in many cases, it has to be assured that some sets of variables are disjoint. For the same reason, the combination of the cases of the structural induction becomes also complicated. The renaming of state variables is used to ensure that all relevant sets of state variables are disjoint. This leads to the theorems `LTL_TO_OMEGA_BASIC_THM` and `LTL_TO_OMEGA_OPTIMIZED_THM`. Addition-

ally, the theorem LTL_TO_OMEGA_OPTIMIZED_THM___IS_LTL_G states that the improved translation of LTL to ω-automata translates a LTL_G formula to a NDET_G automaton formula.

5.5 Translation of PSL to RLTL

To translate PSL to RLTL, some lemmata about PSL and a lot of lemmata about RLTL have to be proved. The most important lemmata about RLTL are presented in Section 3.1. The lemmata about PSL are mostly technical. Compared to the translation presented in Section 3.1, propositional formulas have to be translated additionally, because the propositional formulas of PSL and RLTL are modelled by different HOL types. Apart from these differences, the correctness of the translation of PSL to RLTL is proved as described in Section 3.1. The correctness of the translation of SUFL to RLTL is shown in theorem PSL_TO_RLTL_THM. There are also other theorems that consider clocked statements or paths without special states.

6 Conclusion and Future Work

In this work, an efficient translation of SERE-free FL to LTL and further to ω-automata is presented. This translation is used to identify classes of unclocked SERE-free FL. Furthermore, it has been shown that these classes are as expressive as the corresponding classes of noncounting, deterministic ω-automata. That is the best possible result of an approach that uses a translation to LTL, since the identified classes of PSL are as expressive as the corresponding LTL-classes. In particular, with SUFL_F and SUFL_G, subsets of PSL are syntactically identified that are as expressive as noncounting liveness and safety automata. Moreover, an efficient translation is presented. This is of practical evidence, because these kinds of automata are very useful to handle finite inputs. For example, they can be used for bounded model checking or on the fly validation during simulation.

The translations and class hierarchies presented in this work can directly be used in practice. However, they are not optimal with respect to syntactic sugar. In Section 4.3, this is discussed for the example of the U operator. It is shown that a formula of the form $\varphi \ \mathsf{U} \ \psi$ with $\varphi, \psi \in \mathsf{SUFL}_\mathsf{G}$ does not belong to SUFL_G. However, the formula $\neg\Big(\neg\psi \ \underline{\mathsf{U}} \ (\neg\varphi \wedge \neg\psi)\Big) \in \mathsf{SUFL}_\mathsf{G}$ is on infinite PSL-paths equivalent to $\varphi \ \mathsf{U} \ \psi$ [31]. Furthermore, the algorithm presented in Section 3.3.2 is not able to translate $\varphi \ \mathsf{U} \ \psi$ to a NDET_G automaton, however, $\neg\Big(\neg\psi \ \underline{\mathsf{U}} \ (\neg\varphi \wedge \neg\psi)\Big) \in \mathsf{SUFL}_\mathsf{G}$ is translated to a NDET_G automaton. Therefore, the translations and the definitions of the classes should be extended by cases for the U operator. For other operators defined as syntactic sugar, the situation may be similar. Therefore, all operators defined as syntactic sugar of FL should be investigated in further work. Then, the translations and the definitions of the classes should be extended by cases for operators defined as syntactic sugar. As shown by the example of the U operator, this is important for practice.

This work does not consider SEREs. In general, FL with SEREs cannot be translated to LTL. Therefore, the approach to translate subsets of FL to LTL and further to ω-automata, cannot handle SEREs. However, it is well known how to translate regular expressions to finite state automata [3, 7, 10, 33, 55]. Therefore, a direction for future work is to translate FL with SEREs directly to ω-automata.

Another interesting question for future work is how to handle finite inputs. In [46], a variant of LTL for finite words is introduced. This variant of LTL is translated to a kind of finite automata on finite words. It should be investigated whether this approach can be used to handle SUFL on finite PSL-paths.

Bibliography

[1] Accellera. Property specification language reference manual, version 1.0. http://www.haifa.il.ibm.com/projects/verification/sugar, January 2003.

[2] Accellera. Property specification language reference manual, version 1.1. http://www.eda.org, June 2004.

[3] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Princiles, Techniques, and Tools.* Addison-Wesley, 1986.

[4] ANSI/IEEE Std 1076-1987. *IEEE Standard VHDL Language Reference Manual*. New York, USA, March 1987.

[5] R. Armoni, D. Bustan, O. Kupferman, and M.Y. Vardi. Resets vs. aborts in linear temporal logic. In H. Garavel and J. Hatcliff, editors, *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *LNCS*, pages 65–80, Warsaw, Poland, 2003. Springer.

[6] I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In *Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 363–367, Paris, France, 2001. Springer.

[7] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(1):117–126, 1986.

[8] C. Berthet, O. Coudert, and J.C. Madre. New ideas on symbolic manipulations of finite state machines. In *Conference on Computer Aided Design (ICCD)*, pages 224–227. IEEE, 1990.

[9] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[10] J.A. Brzozowski. Derivates of regular expression. *Journal of the ACM*, 11:481–494, 1964.

[11] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conference (DAC)*, pages 46–51, Orlando, Florida, USA, 1990. IEEE.

[12] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science (LICS)*, pages 1–33, Washington, D.C., June 1990. IEEE Computer Society.

[13] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[14] J.R. Büchi. On a decision method in restricted second order arithmetic. In E. Nagel, editor, *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–12, Stanford, CA, 1960. Stanford University Press.

[15] J.R. Büchi. Weak second order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[16] K.-H. Chang, W.-T. Tu, Y.-J. Yeh, and S.-Y. Kuo. A temporal assertion extension to Verilog. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 3299 of *LNCS*, pages 499–504. Springer, 2004.

[17] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[18] K. Claessen and J. Mårtensson. An operational semantics for weak PSL. In A.J. Hu and A.K. Martin, editors, *Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 3312 of *LNCS*, pages 337–351, Austin, Texas, USA, 2004. Springer.

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

[20] M. Daniele, F. Giunchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In N. Halbwachs and D.A. Peled, editors, *Conference on Computer Aided Verification (CAV)*, volume 1633 of *LNCS*, pages 249–260, Trento, Italy, 1999. Springer.

[21] E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[22] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[23] D.M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Symposium on Principles of Programming Languages (POPL)*, pages 163–173, New York, 1980. ACM.

[24] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 53–65, Paris, France, 2001. Springer.

[25] R. Gerth, D.A. Peled, M.Y. Vardi, and P.L. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification (PSTV)*, Warsaw, June 1995. North Holland.

[26] Mike Gordon. Psl semantics in higher order logic. In *Workshop on Designing Correct Circuits (DCC)*, 2004.

[27] M.J.C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, Computer Laboratory, University of Cambridge, May 1985.

[28] M.J.C. Gordon, J. Hurd, and K. Slind. Executing the formal semantics of the Accellera property specification language by mechanised theorem proving. In D. Geist and E. Tronci, editors, *Conference on Correct Hardware Design and Verification Methods (CHARME)*, volume 2860 of *LNCS*, pages 200–215, L'Aquila, Italy, 2003. Springer.

[29] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[30] M.J.C. Gordon, R. Milner, and C.P. Wadsworth. *A Mechanized Logic of Computation*, volume 78 of *LNCS*. Springer, New York, 1979.

[31] J. Havlicek, D. Fisman, and C. Eisner. Basic results on the semantics of Accellera PSL 1.1 foundation language. Technical Report 2004.02, Accellera, 2004.

[32] J.G. Henriksen, J.L. Jensen, M.E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1019 of *LNCS*, pages 89–110, Aarhus, Denmark, 1995. Springer.

[33] S.C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[34] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Equipment Cooperation, 1991.

[35] L.H. Landweber. Decision problems for ω-automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.

[36] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Symposium on Principles of Distributed Computing*, pages 377–408, 1990.

[37] Z. Manna and A. Pnueli. *The temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[38] N. Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the European Association for Theoretical Computer Science*, 79:122–128, 2003.

[39] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[40] R. McNaughton and S. Papert. *Counter-free Automata*. MIT, 1971.

[41] P. Moorby. History of Verilog. *IEEE Design and Test of Computers*, pages 62–63, September 1992.

[42] A. Pnueli. The temporal logic of programs. In *Symposium on Foundations of Computer Science (FOCS)*, volume 18, pages 46–57, New York, 1977. IEEE Computer Society.

[43] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transaction of the American Mathematical Society*, 141:1–35, 1969.

[44] M.O. Rabin. Automata on infinite objects and Church's problem. In *Regional Conference Series in Mathematics*, volume 13. American Mathematical Society (AMS), 1972.

[45] R. Reetz, K. Schneider, and T. Kropf. Formal specification in VHDL for formal hardware verification. In *Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, February 1998.

[46] J. Ruf, D.W. Hoffmann, T. Kropf, and W. Rosenstiel. Simulation-guided property checking based on multi-valued AR-automata. In *Design, Automation and Test in Europe (DATE)*, Munich, Germany, 2001. IEEE Computer Society.

[47] K. Schneider. *Ein einheitlicher Ansatz zur Unterstützung von Abstraktionsmechanismen der Hardwareverifikation*, volume 116 of *DISKI (Dissertationen zur Künstlichen Intelligenz)*. Infix, Sankt Augustin, 1996. ISBN 3-89601-116-2.

[48] K. Schneider. Improving automata generation for linear temporal logic by considering the automata hierarchy. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 2250 of *LNAI*, pages 39–54, Havanna, Cuba, 2001. Springer.

[49] K. Schneider. *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer, 2003.

[50] K. Schneider and D.W. Hoffmann. A HOL conversion for translating linear time temporal logic to omega-automata. In Y. Bertot, G. Dowek, A. Hirschowitz,

C. Paulin, and L. Théry, editors, *Higher Order Logic Theorem Proving and its Applications (TPHOL)*, volume 1690 of *LNCS*, pages 255–272, Nice, France, 1999. Springer.

[51] T. Schuele and K. Schneider. Bounded model checking of infinite state systems: Exploiting the automata hierarchy. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 17–26, San Diego, CA, June 2004. IEEE.

[52] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In E.A. Emerson and A.P. Sistla, editors, *Conference on Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 248–263, Chicago, IL, USA, 2000. Springer.

[53] R.S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1-2):121–141, 1982.

[54] W. Thomas. *Automata on Infinite Objects*, volume B, chapter Automata on Infinite Objects, pages 133–191. Elsevier, 1990.

[55] Ken Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968.

[56] *IEEE Standard VHDL Language Reference Manual*. New York, USA, June 1993. ANSI/IEEE Std 1076-1993.

[57] J. von Wright. Mechanizing the temporal logic of actions in HOL. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *Higher Order Logic Theorem Proving and its Applications (TPHOL)*, pages 155–159, Davis, California, August 1991. IEEE Computer Society.

[58] K. Wagner. On ω-regular sets. *Information and Control*, 43:123–177, 1979.

[59] P.L. Wolper. Temporal logic can be more expressive. In *Symposium on Foundations of Computer Science (FOCS)*, pages 340–348, New York, 1981. IEEE Computer Society.

[60] P.L. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2):72–99, 1983.

[61] P.L. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computations paths. In *Symposium on Foundations of Computer Science (FOCS)*, pages 185–194, New York, 1983. IEEE Computer Society.