Masaryk University Brno
Faculty of Informatics

Linear Temporal Logic: Expressiveness and Model Checking

Jan Strejček

Ph.D. Thesis

2004

Abstract

Model checking of finite-state systems with specifications given as formulae of Linear Temporal Logic (LTL) is one of the most common verification problems. Like other verification problems, LTL model checking suffers from state explosion. Techniques tackling state explosion usually employ some specific property of the LTL fragment they are designed for. For example, a popular method called partial order reduction is based on the fact that specifications given by LTL formulae without the 'next' operator do not distinguish between the stutter equivalent behaviours of a system.

We study the properties of LTL fragments that are related to model checking. In particular, we are interested in the properties that can potentially lead to new techniques suppressing the state explosion problem. At the same time we study expressiveness and decidability of LTL fragments, and complexity of model checking problem for the fragments. Besides a broad unifying overview of hitherto known results, this thesis presents some original results about LTL fragments with temporal operators 'next' and 'until', where the nesting depths of one or both operators are bounded. More precisely, we extend the above-mentioned stuttering principle to these fragments and describe a new concept of characteristic patterns. In both cases we indicate that our results can improve existing model checking techniques. Furthermore, we develop the established fact that LTL is expressively equivalent to alternating 1-weak Büchi automata (A1W automata). Specifically, we identify the classes of A1W automata that are expressively equivalent to LTL fragments of the until-release hierarchy and LTL fragments using only future temporal operators with or without bounded nesting depths. This thesis also contains a collection of open questions and topics for future work.

Acknowledgements

First of all I would like to thank my supervisor Mojmír Křetínský for his help, encouragement, and many valuable and wide-ranging discussions during my studies. I am grateful to him also for careful reading a draft of this thesis.

Many thanks go to Antonín Kučera for his support and fruitful collaboration. I also much appreciate the collaboration with Radek Pelánek and Vojtěch Řehák.

I thank all the people who have contributed to this thesis in some way. Namely, I thank Michal Kunc for providing the crucial hints which eventually led to the definition of *characteristic patterns*, Nicolas Markey for the clarification of the situation around the *chop* modality, Lenore Zuck for sending me an electronic version of her PhD thesis (at the same time I thank the person who had to scan the whole thesis because of me), Thomas Wilke for sending me a hard copy of his post-doctoral thesis, Ivana Černá and Jiří Srba for providing me with some papers that are neither online nor in our faculty library, and James Thomas for consultation regarding my English.

My thanks go also to all colleagues from Parallel and Distributed Systems Laboratory (ParaDiSe). From time to time they infected me with their almost constant enthusiasm for formal verification.

Finally, I thank my wife Adriana and our daughter Zorka for their love, support, and patience.

Jan Strejček

Contents

Chapter 1

Introduction

Today, the power of computers and sophisticated software methodologies and tools enable the design of very complex systems. Together with an increase in complexity, it is still more and more difficult to make the systems bug-free. Every error found late in the design process can delay the production phase and the consequences of a latent error could be catastrophic. It is extremely important to find all possible bugs and to find them as early in the design process as possible.

Several so-called *design validation methods* have been introduced so far to help developers to find bugs. Two such methods are *testing* and *simulation*. Although they are the oldest ones, they are still heavily used. Both of them are quite effective in early phases of the debugging process, but the effectiveness dramatically decreases when the system hides only a small number of bugs. Moreover, these methods are not usually able to confirm that the design is fully correct as they explore only some of the possible behaviours of the system.

Another approach to design validation is *formal verification*. The main advantage shared by various verification techniques is an ability to pronounce that the system is correct (i.e. the system corresponds to its specification) as they explore all the possible behaviours of the system. There are three basic formal verification techniques.

Theorem proving - The idea is to prove formally that the designed system has the required properties. Unlike the other formal verification techniques, theorem proving has not yet been automatized. However, there are some semi-automatic methods and research into this area is ongoing.

Equivalence checking - Equivalence checking refers to the problem of deciding if a system and its specifications are the same with respect to a given behavioural equivalence or not. Many behavioural equivalences have been suggested and studied within this context. Several

algorithms to solve the equivalence checking problem have been designed and implemented. The main disadvantage of this approach is the need for a detailed specification.

Model checking - Specification is denoted by a formula of suitable modal or temporal logic. Model checking is required to decide whether a given system satisfies the specification formula or not. In the latter case, the algorithms return a negative answer accompanied by a *counterexample*, i.e. a run of the system violating the specification.

These formal verification techniques are aimed at validation of finite-state as well as infinite-state systems. The motivation for verification of finite-state systems is obvious; all concrete instances of hardware and software systems have a finite number of possible states (thanks to the finiteness of data types, limited address space, etc.). However, there are good reasons to deal with verification techniques for infinite-state systems. For example, these techniques can be used for verification of general or parametrized systems. Further, infinite-state abstractions of large finite-state systems can be smaller and the verification of such abstractions can be easier. An example of a natural abstraction that can decrease the complexity is obtained when we forget that in real systems every stack has a limited height.

Many different modal and temporal logics can serve to express the systems specifications for model checking purposes. In general, there are two classes of logics used in this context. The formulae of *linear time* logics are interpreted over linear sequences of actions corresponding to possible runs of the system. The "standard" linear time logic is *Linear Temporal Logic (LTL)*. On the other hand, the formulae of *branching time* logics are interpreted over states (in fact, over computational trees, i.e. the structures where the successors of each state are all states reachable from the state in one step). Unlike the linear case, several branching time logics are used, for example *Computational Tree Logic (CTL)*, *CTL**, and μ*-Calculus*.

One of the most prominent instances of the model checking problem is the problem to decide whether a given finite-state system satisfies specifications expressed by a given formula of the logic LTL. The problem is known as the *LTL model checking of finite-state systems*. In this thesis we restrict our attention just to this particular problem of formal verification.

As commonly agreed, the major disadvantage model checking suffers from is the well-known *state explosion problem*. Roughly speaking, the problem is that the number of global states of a common system is very large in contrast to the length of the system's high level definition (e.g. its source code). This problem has basically two causes. First, the number of global states grows exponentially with the number of parallel components or

threads of the system, which can be high especially in case of communication protocols or complex hardware systems consisting of many separated units. The second cause of state explosion is the large data domains that the common systems are working with. Despite original pessimism, several methods dealing with the state explosion problem have been introduced. Besides *abstraction*, *symbolic model checking*, and *partial order reduction*, which are already well established, there are other promising approaches such as compositional reasoning or distributed model checking.

Abstraction techniques are usually performed on a high level description of the system under verification. Intuitively, abstraction attempts to decrease the size (i.e. the number of global states) of the system by focusing on the aspects that are relevant to a given specification. For example, an abstraction method known as the *cone of influence reduction* eliminates the variables that do not influence the variables in the specification. Further, *data abstraction* transforms the actual data domains into a smaller abstract data domains. Abstraction is often combined with the other methods dealing with state explosion.

Symbolic model checking employs *Ordered Binary Decision Diagrams* (OBDDs – see e.g. [Weg00]) to represent sets of global states of the system. For the systems that are regular in some sense (especially synchronous hardware chips), the OBDDs are much smaller than the representation by explicit enumeration. This fact and the existence of fast algorithms for manipulation with OBDDs have led to the development of many academic and even some commercial symbolic model checkers, which are used with satisfactory success for verifying hardware circuits. Unfortunately, the state spaces of software systems are usually "less regular" than state spaces of hardware systems. Therefore, the representation by OBDDs is not efficient enough in this case.

Another few promising techniques tackling state explosion have been developed for LTL model checking. However, these techniques usually do not work for a general LTL formulae. In fact, they employ the specific properties of LTL fragments they are designed for. For example, partial order reduction methods can be used when a specification formula does not contain the temporal operator *next* saying that 'a subformula holds in the next state' – these methods employ the fact that system specifications given by such a formula are not sensitive to an order of independent actions (two action are independent when executing them in either of the order results in the same global state). Other examples are efficient algorithms for model checking of *safety* formulae, i.e. formulae saying that there is no reachable state with particular characteristics. We believe that the research on properties of LTL fragments can lead to new and more efficient model checking techniques.

1.1 Subject of this thesis

The central subject of this thesis are fragments of LTL and their attributes relevant to model checking. More precisely, we are interested in the following properties.

Expressiveness – The expressiveness of an LTL fragment is measured by languages corresponding to the formulae of the fragment. We study various characterizations of these language classes in terms of first order logic, finite automata, regular expressions, and algebraic structures. The relative expressiveness of LTL fragments is studied as well.

Decidability – A fragment is said to be *decidable* if there is a procedure deciding whether a given language can be defined by a formula of the fragment or not.

Complexity of model checking – We study the asymptotic complexity of the considered model checking problem and its subproblem called *model checking a path* [MS03].

Other properties – We are also searching for all properties of LTL fragments that can potentially improve efficiency or tractability of model checking.

We have already provided a motivation for the properties in the last item of the list given above. The motivation for research on expressiveness, decidability, and complexity of model checking is straightforward. Let us assume that we want to decide whether a finite-state system meets a given specification. In this situation we have to deal with the questions like: In what fragments of LTL can the specification be denoted? What is the complexity of model checking for these fragments? Are there any techniques improving efficiency of model checking (like specialized model checking algorithms or partial order reduction methods) available for these fragments? Another natural question is whether the specification is satisfiable. This is why we also study the complexity of the satisfiability problem.

1.2 Thesis contribution

Research in the area of LTL fragments has been vital during the last decade. There are several papers surveying the results achieved, but they usually concentrate on a specific aspect of the topic only. For example, the paper [Wil99] is focused on expressiveness and decidability of LTL fragments. Moreover, it is already out of date. A nice overview of complexity of the model checking problem for various fragments of LTL, CTL, and CTL* written by Schnoebelen [Sch03] concentrates on the asymptotic complexity of the problem. To the best of author's knowledge, there is no paper

summarizing the properties of LTL fragments that lead to (asymptotic or nonasymptotic) improvements in model checking.

The contribution of this thesis can be divided into two parts.

- This thesis provides a broad and unifying summary of results regarding LTL fragments and their relations to model checking discussed in the previous section. This thesis includes
  - a comprehensive overview of temporal modalities and various LTL fragments considered in literature,
  - a summary of results on expressiveness and the decidability of these fragments, and
  - a survey of complexity results for satisfiability and model checking problems for LTL fragments.
- Further, this thesis presents the following original results. Most of them are connected with LTL fragments that contain only widespread future modalities *next* and *until*, and where the nesting depth of *next* is bounded by parameter n (denoted $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$), or nesting depths of *until* is bounded by parameter m (denoted $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$), or both nesting depths are bounded (denoted $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$).
  - We formulate extended versions of the well-known *stuttering principle*. Using these extended principles we show that hierarchies of the fragments are strict and we also indicate that the principles can improve partial order reduction methods.
  - We introduce a concept of *characteristic patterns*. This concept provides new insights into the expressive power of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragments. Further, we suggest three application of characteristic patterns that can potentially improve existing model checking techniques.
  - It is known that *alternating 1-weak (A1W) automata* have the same expressive power as LTL. We improve the translation of A1W automata to equivalent LTL formulae. The improved translation allows us to characterize the language classes corresponding to LTL fragments mentioned above and also to fragments with the temporal operator *eventually*. We also provide a characterization of language classes corresponding to fragments of the *until-release hierarchy* [ČP03].
  - Finally, few original (and almost evident) statements are included directly in the overview of known expressiveness results and in the survey of known complexity results.

1.3 Thesis organization

The thesis is organized as follows.

Chapter 2 recalls all formalisms employed in the thesis, namely LTL and its fragments, first-order monadic logic and its fragments, and selected parts of automata and formal languages theory. We also describe a classic automata-based algorithm for LTL model checking, theoretical background of standard partial order reduction methods, and a subproblem called *model checking a path*. The definition of LTL is divided into two parts. First we define the "light version" using only future modalities *next* and *until*. This version is sufficient for almost all the original results presented in Chapters 5, 6, and 7. Then we present an overview of other modalities occurring in literature accompanied by basic relations between the modalities.

Chapter 3 summarizes known expressiveness and decidability results for LTL fragments. Further, this chapter contains a section devoted to the succinctness of LTL fragments with respect to other equivalent formalisms. In this chapter we also mention many fragment properties, like standard stuttering principle or relations to automata of special type, that can lead to improvements of model checking algorithms.

Chapter 4 provides a survey of results concerning the asymptotic complexity of satisfiability, model checking, and model checking a path problems for various LTL fragments.

Chapter 5 extends the standard stuttering principle to more general *letter stuttering*, *subword stuttering*, and *general stuttering* principles. These new principles allow us to prove the semantic strictness of natural hierarchies of $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragments. Further, we provide an effective characterization of languages definable by $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ formulae. We also indicate potential applications of letter stuttering and general stuttering in partial order reduction.

Chapter 6 gives a new characterization of languages that are definable in fragments of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. We also propose a generic method for decomposing LTL formulae into an equivalent disjunction of "semantically refined" LTL formulae, and indicate how this result can be used to improve the functionality of existing LTL model checkers. Application of characteristic patterns in model checking a path and partial order reduction are suggested as well.

Chapter 7 reviews the translations between LTL and A1W automata showing that the two formalisms are expressively equivalent. We improve

the translation of A1W automata to LTL formulae. Using the improved translation we identify the classes of A1W automata equivalent to $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragments, and to those fragments extended with the temporal operator *eventually*. Further, we identify the classes of A1W automata equivalent to fragments of the until-release hierarchy [ČP03].

Chapter 8 recapitulates the content of the thesis and the most important topics for future work.

The last sections of Chapters 3–7 are called "Additional notes". These sections contain many open questions and topics for future work related to the subject of the corresponding chapter.

1.4 Author's publications

This section lists all publications (co-)authored by the author of this thesis.

Journal articles

1. Antonín Kučera and Jan Strejček. *The Stuttering Principle Revisited.* Acta Informatica. To appear.

International conference and workshop proceedings

1. Antonín Kučera and Jan Strejček. *Characteristic Patterns for LTL.* In SOFSEM 2005, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.

2. Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. *Extended Process Rewrite Systems: Expressiveness and Reachability.* In Philippa Gardner and Nobuko Yoshida, editors, CONCUR 2004 - Concurrency Theory, volume 3170 of Lecture Notes in Computer Science, pages 355–370. Springer-Verlag, 2004.

3. Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. *On Extensions of Process Rewrite Systems: Rewrite Systems with Weak Finite-State Unit.* In Philippe Schnoebelen, editor, INFINITY 2003: 5th International Workshop on Verification of Infinite-State Systems, volume 98 of Electronic Notes in Theoretical Computer Science, pages 75–88. Elsevier Science Publishers, 2004.

4. Antonín Kučera and Jan Strejček. *The Stuttering Principle Revisited: On the Expressiveness of Nested X and U Operators in the Logic LTL.* In Julian Bradfield, editor, CSL 2002: 11th Annual Conference of the

European Association for Computer Science Logic, volume 2471 of Lecture Notes in Computer Science, pages 276–291. Springer-Verlag, 2002.

5. Jan Strejček. *Rewrite Systems with Constraints.* In Luca Aceto and Prakash Panangaden, editors, EXPRESS 2001: 8th International Workshop on Expressiveness in Concurrency, volume 52 of Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2002.

Other work

1. Jitka Crhová, Pavel Krčál, Jan Strejček, David Šafránek, and Pavel Šimeček. *YAHODA: Verification Tools Database.* In Ivana Černá, editor, Tools Day, Proceedings, FIMU-RS-2002-05, pages 99–103. Faculty of Informatics, Masaryk University Brno, 2002.

2. Jan Strejček. *Boundaries and Efficiency of Verification.* In Proceedings of summer school on MOdelling and VErification of Parallel processes (MOVEP 2002), pages 403–408. IRCCyN, Ecole Centrale de Nantes, 2002.

Technical reports

Six technical reports, mostly extended versions of the conference and workshop papers.

The results presented in journal article 1 and conference paper 4 form the base Chapter 5 while the results from conference paper 1 can be found in Chapter 6.

The other conference and workshop papers deal with extended *process rewrite systems*, their expressiveness, and the reachability problem for the extended systems. The results from this area are not included here as a mono-thematic thesis is preferred.

Chapter 2

Preliminaries

This chapter recalls the formalisms used in the thesis. We provide definition of linear temporal logic including a comprehensive survey of temporal operators introduced so far, definition of first-order monadic logic of order, and selected parts of formal languages and automata theory. Finally, we mention the model checking problem and describe some aspects of model checking area related to our topic.

First of all we establish the following convention. The set of natural numbers with zero is denoted by $\mathbb{N}_0$, while the set of natural numbers without zero is denoted by $\mathbb{N}$.

2.1 Linear temporal logic

The genesis of linear temporal logic is linked with *modal logic*. Modal logic has been developed by philosophers to study reasoning that involves the use of *modalities*, i.e. expressions of the form "it is possible that ..." and "it is necessary that ...". Modal logic is interpreted over a set of *possible worlds*. The truth values of atomic propositions are determined by the considered world. Besides the boolean connectives, the logic uses the unary *modal* operators (or *modalities*) *possibly* and *necessarily* denoting that a proposition is true in some possible world or all possible worlds of the considered set respectively. These operators have been already used (also in temporalized form) by ancient Greek philosophy schools and further refined into different variants of necessity and possibility by European and Arabian logicians during Middle Ages.

The twentieth century brought new wave of interest in modal logics. In 1963, Kripke [Kri63] formalized the semantics of modal logic based on the idea of possible worlds. Even before that, Prior [Pri57] introduced the temporal variant of modal logic under the name of *Tense Logic*; a set of possible worlds is ordered into a (temporal) sequence and the modal operators *possibly* and *necessarily* become the temporal operators *eventually* and *always*.

The variety of other modalities have been proposed during the following years, particularly the binary modalities *until* and *since* (presented by Kamp in [Kam68]) and the unary modality *next*.

Many temporal logics have been introduced so far. They can be classified with respect to the considered model of time flow as *linear time* or *branching time*. In linear time concept, each instant has only one possible future. That is, a set of possible worlds is ordered into a linear sequence. In branching time logics, each instant may have several distinct futures (but only one history). In this case, possible worlds are ordered into a tree structure. The concept of branching time naturally corresponds to the non-determinism. However, non-determinism can be modelled in linear time as well; instead of one linear sequence we can consider a set of linear sequences, each one corresponding to a particular course of events.

The best way to document the potential of temporal logic framework is to mention some of main applications. First applications of temporal logics can be found in the area of natural language analysis. The framework has been also accepted by artificial intelligence community for dealing with temporal issues. A concrete application in computer science was proposed for the first time by Burstall in 1974 [Bur74]. His approach has been elaborated by Pnueli in [Pnu77]. The paper is now considered to be the classic source of program specification and verification based on temporal logic.

A popular examples of branching time temporal logic used in the context of computer science are the *Computation Tree Logic (CTL)* introduced by Clarke and Emerson in [CE81] and the *CTL** logic introduced by Emerson and Halpern in [EH86]. No less popular example of linear time temporal logic is the *Linear Temporal Logic (LTL)* [Pnu77, GPSS80] – the central subject of the thesis.

First of all, we should specify a variant of LTL we deal with in the thesis. As written above, LTL (as a linear time logic) is interpreted over (sets of) linear sequences of instants. However, this still allows several distinct models of time flow. For example, the linear sequence can be discrete or continuous. At the same time, it can be bounded on both side or just on one side (the beginning of the sequence is usually given in this case) or unbounded at all. In the following we work with (finite or infinite) discrete linear sequences with a given beginning. This is a natural choice in the context of computer science as the beginning of a linear sequence corresponds to the launch of a studied system (e.g. software or hardware) that can terminate after a finite number of steps or run forever.

More or less independently on a model of time flow, one can consider various extensions of LTL. Here we mention just two important examples as LTL extensions are out of the scope of the thesis. One of the oldest LTL extensions is *Extended Temporal Logic (ETL)* defined by Wolper in [Wol83]. This extension enables LTL to express properties definable by right-linear

grammars. Other LTL extensions have been designed to deal with real time. For more details about timed logics we refer to the survey [AH92].

2.1.1 Basic syntax and semantics

The linear temporal logic can be equivalently defined with use of various sets of temporal operators. In the following definition, we employ just two common temporal operators, namely *next* and *until*. We choose these modalities as they play the main role in Chapters 5, 6, and 7 devoted to results of our research. Later we introduce the other temporal operators and argue that they do not change the expressive power of the logic.

The syntax of LTL formulae (ranged over by $\varphi, \psi, \ldots$) is given by the following abstract syntax:

$$\varphi \ ::= \ \top \ \mid \ p \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}\varphi \ \mid \ \varphi \,\mathsf{U}\, \varphi,$$

where

- $\top$ stands for *true*,
- p ranges over a countable set of *atomic propositions* $At = \{o, p, q, \ldots\}$,
- $\neg$ and $\wedge$ are boolean operators *negation* and *conjunction* respectively,
- X and U are temporal operators *next* and *until* respectively.

Given a formula φ, $|\varphi|$ denotes its length and $At(\varphi)$ denotes the set of atomic propositions occurring in φ.Other boolean connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are derived operators defined via the standard abbreviations. We also use $\bot$ (*false*), $\mathsf{F}\varphi$ (*eventually)*, $\mathsf{G}\varphi$ (*always*), and $\varphi \,\mathsf{R}\, \psi$ (*release*) to abbreviate $\neg\top$, $\top \,\mathsf{U}\, \varphi$, $\neg(\top \,\mathsf{U}\, \neg\varphi)$, and $\neg(\neg\varphi \,\mathsf{U}\, \neg\psi)$ respectively.

In order to reduce the number of parentheses in formulae we establish a priority order of the operators as follows.

- Unary operators ($\neg, \mathsf{X}, \mathsf{F}, \mathsf{G}$, and operators defined later) have higher priority than all binary operators.
- Binary temporal operators (U, R, and temporal operators defined later) have higher priority than $\wedge, \vee, \Rightarrow, \Leftrightarrow$.
- $\wedge, \vee$ have higher priority than $\Rightarrow, \Leftrightarrow$.
- $\Rightarrow$ has higher priority than $\Leftrightarrow$.

Example 2.1 *We write* $\mathsf{X}p \ \mathsf{U}\, q \Rightarrow \neg o \vee \mathsf{F}q$ *instead of* $((\mathsf{X}p) \,\mathsf{U}\, q) \Rightarrow ((\neg o) \vee \mathsf{F}q)$.

We define the semantics of LTL in terms of (languages over) infinite or nonempty finite words. An *alphabet* is a set $\Sigma_P = 2^P$, where $P \subseteq At$ is a finite set of atomic propositions. Elements of an alphabet are called *letters*. A finite sequence of letters from an alphabet Σ is called *finite word* or *string* over Σ. An infinite sequence of letters from an alphabet Σ is called *infinite word* or *ω-word* over Σ. By *word* we refer to both finite and infinite words. A set of words over Σ is called *language* over Σ. Languages of ω-words are sometimes called *ω-languages*. Languages of all strings, nonempty strings, and ω-words over an alphabet Σ are denoted by Σ^*, Σ^+, and Σ^ω respectively. We use $a, b, c \ldots$ to range over Σ, $u, v, w, \ldots$ to range over Σ^*, and $\alpha, \beta, \ldots$ to range over Σ^ω. Finally, we use π to denote a finite or infinite word.

Given a word π, by $|\pi|$ we denote the length of π (for infinite words we set $|\pi| = \infty$). An empty word ε has a zero length. A concatenation of a string u and a word π is denoted by $u.\pi$ or $u\pi$. For all $0 \leq i < |\pi|$ by $\pi(i)$ we denote the $(i{+}1)^{th}$ letter of π, i.e. $\pi = \pi(0)\pi(1)\pi(2)\ldots\pi(|\pi|-1)$ if π is finite and $\pi = \pi(0)\pi(1)\pi(2)\ldots$ otherwise. Further, for all $0 \leq i < |\pi|$ by π_i we denote the i^{th} suffix of π, i.e. $\pi_i = \pi(i)\pi(i{+}1)\pi(i{+}2)\ldots\pi(|\pi|{-}1)$ if π is finite and $\pi_i = \pi(i)\pi(i{+}1)\pi(i{+}2)\ldots$ otherwise. Moreover, for all $0 \leq i < |\pi|$ and $j \geq 1$ such that $i + j - 1 < |\pi|$ the symbol $\pi(i, j)$ denotes the subword of π of length j which starts with $\pi(i)$, i.e. $\pi(i, j) = \pi(i)\pi(i+1)\ldots\pi(i+j-1)$. A *pointed word* is a pair (π, i) of a nonempty word π and a *position* $0 \leq i < |\pi|$ in this word.

Let φ be an LTL formula and (π, i) be a pointed word. We define when a formula φ is *valid* for the position i in the word π, written $(\pi, i) \models \varphi$, by induction on the structure of φ.

$$
\begin{array}{lll}
(\pi,i) \models \top & & \\
(\pi,i) \models p & \text{iff} & p \in \pi(i) \\
(\pi,i) \models \neg\varphi & \text{iff} & (\pi,i) \not\models \varphi \\
(\pi,i) \models \varphi \wedge \psi & \text{iff} & (\pi,i) \models \varphi \ \wedge \ (\pi,i) \models \psi \\
(\pi,i) \models \mathsf{X}\varphi & \text{iff} & i+1 < |\pi| \ \wedge \ (\pi, i+1) \models \varphi \\
(\pi,i) \models \varphi \,\mathsf{U}\, \psi & \text{iff} & \exists k.\ (i \leq k < |\pi| \ \wedge \ (\pi,k) \models \psi \ \wedge \\
 & & \qquad \wedge\ \forall j.\ (i \leq j < k \ \Rightarrow\ (\pi,j) \models \varphi))
\end{array}
$$

Intuitively, $\mathsf{X}\varphi$ says that φ is true in the next position (and this next position exists). A formula $\varphi \,\mathsf{U}\, \psi$ means that ψ is true either now or in the future and φ holds since now until that moment.

We say that a word π *satisfies* φ, written $\pi \models \varphi$, if $(\pi, 0) \models \varphi$. Given an alphabet Σ, an LTL formula φ defines the languages of finite or infinite

words over Σ in the following way:

$$L_F^\Sigma(\varphi) = \{u \in \Sigma^+ \mid u \models \varphi\}$$
$$L^\Sigma(\varphi) = \{\alpha \in \Sigma^\omega \mid \alpha \models \varphi\}$$

We omit the superscript Σ if $\Sigma = 2^{At(\varphi)}$.

On the other hand, a language L (of finite or infinite words) over an alphabet Σ is said to be *LTL language* or *LTL property* if there exists an LTL formula φ satisfying $L = L_F^\Sigma(\varphi)$ or $L = L^\Sigma(\varphi)$.

The definition of LTL languages indicates that we work with languages of finite words and with languages of infinite words, but not with languages of finite and infinite words together. In this thesis we prefer to work with infinite words. However, we do not restrict our attention only to infinite words as some interesting results about LTL (e.g. characterization of LTL fragments via forbidden patterns) valid for finite words have no proper counterparts in terms of infinite words. In the following chapters the logic is interpreted over infinite words unless stated otherwise.

Contrary to the model checking area, where the presented definition of LTL fits perfectly, for theoretical study of LTL properties it is often more convenient to work with the variant of LTL using *letters* instead of atomic propositions. More precisely, in syntax of the logic the atomic propositions are replaced by letters and the second line in the definition of LTL semantics is replaced by

$$(\pi, i) \models a \quad \text{iff} \quad a = \pi(i).$$

We use $a, b, c, \ldots$ to range over letters. The definition of LTL based on letters is also used in Chapters 5, 6, and 7 presenting the original results. The difference between the two presented versions of LTL is very subtle and results proven for one setting can be easily adapted to the other.

2.1.2 Satisfiability, validity, and equivalence

The decision whether we consider all pointed words or just the pointed words of the form $(\pi, 0)$ yields different notions of satisfiability, validity, and equivalence.

Definition 2.2 *A formula φ is said to be* (initially) satisfiable *if there exists a word π such that $\pi \models \varphi$. A formula φ is said to be* globally satisfiable *if there exists a word π and a position i such that $(\pi, i) \models \varphi$.*

Definition 2.3 *A formula φ is said to be* (initially) valid *if for all words π we have $\pi \models \varphi$. A formula φ is said to be* globally valid *if for all words π and for all positions $0 \leq i < |\pi|$ we have $(\pi, i) \models \varphi$.*

It is easy to see that if a formula is initially satisfiable then it is also globally satisfiable. On the other hand, if a formula is globally valid, then it is also initially valid. Moreover, a formula is globally satisfiable if and only if its negation is not globally valid. Analogous proposition holds for initial versions of satisfiability and validity.

Definition 2.4 *Let φ and ψ be LTL formulae. The formulae are said to be* (initially) equivalent, *written $\varphi \equiv_i \psi$, if for all words π we have*

$$\pi \models \varphi \text{ if and only if } \pi \models \psi.$$

The formulae are said to be globally equivalent, *written $\varphi \equiv_g \psi$, if for all words π and for all positions $0 \leq i < |\pi|$ we have*

$$(\pi, i) \models \varphi \text{ if and only if } (\pi, i) \models \psi.$$

One can readily confirm that if two formulae are globally equivalent then they are also initially equivalent.

Let us mention that there is no difference between initial and global versions of satisfiability, validity, and equivalence as far as we consider only so called future modalities like until and next. Indeed, for all pointed words (π, i) and all formulae φ containing only these modalities it holds that

$$(\pi, i) \models \varphi \text{ if and only if } \pi_i \models \varphi$$

and hence the respective initial and global versions coincide. However, the difference between initial and global versions arises as soon as past modalities are introduced.

The satisfiability, validity, and equivalence problems are often considered in context of either finite or infinite words only. However, the algorithms solving these problems for infinite words can be usually transformed to solve the problems for finite words with the same complexity.

2.1.3 Other temporal operators

In this subsection we enrich the syntax of LTL with other modalities considered in the literature. For the sake of completeness, presented list of modalities includes the operators already defined as well. The descriptions of individual operators contain alternative names and notations, formal semantics, and basic relations between modalities. These relations represent the first step in our study of expressive power of temporal operators.

Let us note that we present only temporal operators that we have found in LTL related literature. Of course, one can define other temporal operators like e.g. strict release or past versions of all the presented future operators. We do not do this as the definition of new temporal operators is not the goal of this thesis.

All temporal modalities can be divided into three categories, namely future modalities, past modalities, and other modalities.

Future modalities

A unary temporal operator M is said to be *future operator* if the validity of a formula $M\varphi$ for a pointed word (π, i) is determined by validity of the subformula φ for pointed words of the form (π, j) where $i \leq j$. The definition of binary future modalities is analogous. It is easy to see that both X and U are future modalities. In literature dealing with LTL one can encounter the following future modalities.

$\mathsf{X}\varphi$ – The operator called *next* or *nexttime*. It is sometimes denoted as $\bigcirc$ or $\mathsf{X}_\exists$.

$\mathsf{X}_\mathsf{w}\varphi$ – The operator *weak next*. It is sometimes denoted as $\mathsf{X}_\forall$. While $\mathsf{X}\varphi$ is never valid for the last position in a finite word, $\mathsf{X}_\mathsf{w}\varphi$ is valid in this case.

$$(\pi, i) \models \mathsf{X}_\mathsf{w}\varphi \quad \text{iff} \quad i + 1 < |\pi| \;\Rightarrow\; (\pi, i + 1) \models \varphi$$

The operator is dual to X operator:

$$\begin{aligned} \mathsf{X}\varphi &\equiv_g \neg\mathsf{X}_\mathsf{w}\neg\varphi \\ \mathsf{X}_\mathsf{w}\varphi &\equiv_g \neg\mathsf{X}\neg\varphi \end{aligned}$$

$\varphi\,\mathsf{U}\,\psi$ – The binary operator called *until* or *strong until*. It is sometimes denoted as $\mathsf{U}_\exists$. The same name is sometimes used for the two following versions of until operator.

$\varphi\,\mathsf{U}_\mathsf{s}\,\psi$ – The operator *strict until* is sometimes denoted as $\mathsf{U}^>$ or $\mathsf{X}\mathsf{U}$. The strictness means that the validity of $\varphi\,\mathsf{U}_\mathsf{s}\,\psi$ for a pointed word depends on the validity of the subformulae φ and ψ for strictly future positions in the word.

$$\begin{aligned} (\pi, i) \models \varphi\,\mathsf{U}_\mathsf{s}\,\psi \quad \text{iff} \quad \exists k.\, (i < k < |\pi| \;\wedge\; (\pi, k) \models \psi \;\wedge \\ \wedge\; \forall j.\, (i < j < k \;\Rightarrow\; (\pi, j) \models \varphi)) \end{aligned}$$

Both X and U operators can be expressed using U_s. Further, U_s can be expressed using the operators X and U:

$$\begin{aligned} \mathsf{X}\varphi &\equiv_g \bot\,\mathsf{U}_\mathsf{s}\,\varphi \\ \varphi\,\mathsf{U}\,\psi &\equiv_g \psi \vee (\varphi \wedge \varphi\,\mathsf{U}_\mathsf{s}\,\psi) \\ \varphi\,\mathsf{U}_\mathsf{s}\,\psi &\equiv_g \mathsf{X}(\varphi\,\mathsf{U}\,\psi) \end{aligned}$$

Hence, LTL can be equivalently defined using just one temporal operator U_s.

$\varphi \,\mathsf{U_w}\, \psi$ – The operator called *weak until* or *waiting-for* or *unless* is sometimes denoted as W or $\mathsf{U}_\forall$. In contrast to $\varphi \,\mathsf{U}\, \psi$, a formula $\varphi \,\mathsf{U_w}\, \psi$ does not say that ψ must be true now or in the future.

$$\begin{aligned}(\pi, i) \models \varphi \,\mathsf{U_w}\, \psi \quad \text{iff} \quad & \forall j.\ (i \leq j < |\pi| \ \Rightarrow\ (\pi, j) \models \varphi) \ \vee \\ & \vee\ \exists k.\ (i \leq k < |\pi| \ \wedge\ (\pi, k) \models \psi \ \wedge \\ & \qquad \wedge\ \forall j.\ (i \leq j < k \ \Rightarrow\ (\pi, j) \models \varphi))\end{aligned}$$

The operator is expressively equivalent to U operator:

$$\begin{aligned}\varphi \,\mathsf{U}\, \psi \quad &\equiv_g \quad \varphi \,\mathsf{U_w}\, \psi \ \wedge\ \neg(\neg\psi \,\mathsf{U_w}\, \bot) \\ \varphi \,\mathsf{U_w}\, \psi \quad &\equiv_g \quad \varphi \,\mathsf{U}\, \psi \ \vee\ \neg(\top \,\mathsf{U}\, \neg\varphi)\end{aligned}$$

By analogy, the strict version of this operator defined in [Krö87] under the name *unless* is equivalent to $\mathsf{U_s}$.

$\varphi \,\mathsf{R}\, \psi$ – The operator *release*. Informally, $\varphi \,\mathsf{R}\, \psi$ means that ψ remains true forever or at least to the position where φ is true (including this position).

$$\begin{aligned}(\pi, i) \models \varphi \,\mathsf{R}\, \psi \quad \text{iff} \quad & \forall j.\ (i \leq j < |\pi| \ \Rightarrow\ (\pi, j) \models \psi) \ \vee \\ & \vee\ \exists k.\ (i \leq k < |\pi| \ \wedge\ (\pi, k) \models \varphi \wedge \psi \ \wedge \\ & \qquad \wedge\ \forall j.\ (i \leq j < k \ \Rightarrow\ (\pi, j) \models \psi))\end{aligned}$$

The operator is dual to U operator:

$$\begin{aligned}\varphi \,\mathsf{U}\, \psi \quad &\equiv_g \quad \neg(\neg\varphi \,\mathsf{R}\, \neg\psi) \\ \varphi \,\mathsf{R}\, \psi \quad &\equiv_g \quad \neg(\neg\varphi \,\mathsf{U}\, \neg\psi)\end{aligned}$$

$\varphi \trianglelefteq \psi$ – The operator *as long as* introduced by Lamport [Lam83a]. A formula $\varphi \trianglelefteq \psi$ says that ψ remains true at least as long as φ does.

$$\begin{aligned}(\pi, i) \models \varphi \trianglelefteq \psi \quad \text{iff} \quad & \forall k.\ (i \leq k < |\pi| \ \Rightarrow \\ & \qquad \Rightarrow\ ((\pi, k) \models \psi \ \vee \\ & \qquad\quad \vee\ \exists j.\ (i \leq j \leq k \ \wedge\ (\pi, j) \models \neg\varphi)))\end{aligned}$$

The operator is expressively equivalent to U operator:

$$\begin{aligned}\varphi \,\mathsf{U}\, \psi \quad &\equiv_g \quad \neg\psi \trianglelefteq \varphi \\ \varphi \trianglelefteq \psi \quad &\equiv_g \quad \psi \,\mathsf{U}\, \neg\varphi\end{aligned}$$

The strict version of this operator (with the arguments swapped) has been introduced under the name *while* in [Krö87]. It is equivalent to $\mathsf{U_s}$ operator.

$\varphi \,\mathsf{B}\, \psi$ – The operator called *before* or *precedes*. Intuitively, $\varphi \,\mathsf{B}\, \psi$ means that if ψ is true now or in the future then φ is true in some position before that moment.

$$(\pi, i) \models \varphi \,\mathsf{B}\, \psi \quad \text{iff} \quad \forall k.\, ((i \leq k < |\pi| \ \wedge\ (\pi, k) \models \psi) \ \Rightarrow \\ \Rightarrow \ \exists j.\, (i \leq j < k \ \wedge\ (\pi, j) \models \varphi))$$

The operator is expressively equivalent to R operator and thus also to U operator:

$$\begin{aligned} \varphi \,\mathsf{R}\, \psi &\equiv_g \varphi \,\mathsf{B}\, \neg\psi \\ \varphi \,\mathsf{B}\, \psi &\equiv_g \varphi \,\mathsf{R}\, \neg\psi \end{aligned}$$

By analogy, the strict version of this operator defined in [Krö87] is equivalent to $\mathsf{U_s}$.

$\varphi \,\mathsf{A}\, \psi$ – The operator *at next* or *first time* introduced in [Krö87]. A formula $\varphi \,\mathsf{A}\, \psi$ says that either ψ is never true in the future or φ is true in the first position where ψ is true.

$$(\pi, i) \models \varphi \,\mathsf{A}\, \psi \quad \text{iff} \quad \forall k.\, (i < k < |\pi| \ \Rightarrow\ (\pi, k) \models \neg\psi) \ \vee \\ \vee \ \exists k.\, (i < k < |\pi| \ \wedge\ (\pi, k) \models \psi \wedge \varphi \ \wedge \\ \wedge\ \forall j.\, (i < j < k \ \Rightarrow\ (\pi, j) \models \neg\psi))$$

The operator is expressively equivalent to $\mathsf{U_s}$ operator:

$$\begin{aligned} \varphi \,\mathsf{U_s}\, \psi &\equiv_g \psi \,\mathsf{A}\, (\psi \vee \neg\varphi) \ \wedge\ \neg(\bot \,\mathsf{A}\, \psi) \\ \varphi \,\mathsf{A}\, \psi &\equiv_g \neg\psi \,\mathsf{U_s}\, (\varphi \wedge \psi) \ \vee\ \neg(\top \,\mathsf{U_s}\, \psi) \end{aligned}$$

$\mathsf{F}\varphi$ – The operator called *eventually* or *sometime (in the future)*. It is sometimes denoted as $\Diamond$. Informally, $\mathsf{F}\varphi$ means that φ is true now or in some future position.

$$(\pi, i) \models \mathsf{F}\varphi \quad \text{iff} \quad \exists k.\, (i \leq k < |\pi| \ \wedge\ (\pi, k) \models \varphi)$$

As already mentioned, the operator can be expressed using U operator:

$$\mathsf{F}\varphi \equiv_g \top \,\mathsf{U}\, \varphi$$

$\mathsf{G}\varphi$ – The operator called *always (in the future)* or *globally* or *henceforth*. It is sometimes denoted as $\Box$. A formula $\mathsf{G}\varphi$ means that φ is true now as well as in all future positions.

$$(\pi, i) \models \mathsf{G}\varphi \quad \text{iff} \quad \forall k.\, (i \leq k < |\pi| \ \Rightarrow\ (\pi, k) \models \varphi)$$

The operator is dual to F operator:

$$\begin{aligned} \mathsf{F}\varphi &\equiv_g \neg\mathsf{G}\neg\varphi \\ \mathsf{G}\varphi &\equiv_g \neg\mathsf{F}\neg\varphi \end{aligned}$$

$\mathsf{F}_s\varphi$ – The operator *strict eventually* is sometimes denoted as $\mathsf{F}^>$ or XF. Intuitively, $\mathsf{F}_s\varphi$ means that φ is true in some future position.

$$(\pi, i) \models \mathsf{F}_s\varphi \quad \text{iff} \quad \exists k.\, (i < k < |\pi| \ \wedge \ (\pi, k) \models \varphi)$$

The operator can be expressed using the operator U_s or the operators F and X. Further, the operator F can be expressed with F_s operator:

$$\begin{aligned} \mathsf{F}_s\varphi &\equiv_g \top\, \mathsf{U}_s\, \varphi \\ \mathsf{F}_s\varphi &\equiv_g \mathsf{XF}\varphi \\ \mathsf{F}\varphi &\equiv_g \varphi \vee \mathsf{F}_s\varphi \end{aligned}$$

$\mathsf{G}_s\varphi$ – The operator *strict always* is sometimes denoted as $\mathsf{G}^>$ or XG. A formula $\mathsf{G}_s\varphi$ says that φ is true in all future positions.

$$(\pi, i) \models \mathsf{G}_s\varphi \quad \text{iff} \quad \forall k.\, (i < k < |\pi| \ \Rightarrow \ (\pi, k) \models \varphi)$$

The operator is dual to F_s operator:

$$\begin{aligned} \mathsf{F}_s\varphi &\equiv_g \neg\mathsf{G}_s\neg\varphi \\ \mathsf{G}_s\varphi &\equiv_g \neg\mathsf{F}_s\neg\varphi \end{aligned}$$

$\overset{\infty}{\mathsf{F}}\varphi$ – The operator *infinitely often*. A formula $\overset{\infty}{\mathsf{F}}\varphi$ interpreted over an infinite word says that φ is true in infinitely many future positions. The same formula interpreted over a finite word means that φ is true at least in the last position in the word. Hence, the operator is sometimes called *finally* in the context of finite words.

$$\begin{aligned} (\pi, i) \models \overset{\infty}{\mathsf{F}}\varphi \quad \text{iff} \quad \forall k.\, (i \leq k < |\pi| \ \Rightarrow \\ \Rightarrow \ \exists j.\, (k \leq j < |\pi| \ \wedge \ (\pi, j) \models \varphi)) \end{aligned}$$

The operator can be expressed using F operator:

$$\overset{\infty}{\mathsf{F}}\varphi \equiv_g \mathsf{GF}\varphi \equiv_g \neg\mathsf{F}\neg\mathsf{F}\varphi$$

$\overset{\infty}{\mathsf{G}}\varphi$ – The operator called *almost always* or *almost everywhere*. A formula $\overset{\infty}{\mathsf{G}}\varphi$ interpreted over an infinite word means that φ is true in all but a finite number of positions. The formula interpreted over a finite word has the same meaning as $\overset{\infty}{\mathsf{F}}\varphi$.

$$\begin{aligned} (\pi, i) \models \overset{\infty}{\mathsf{G}}\varphi \quad \text{iff} \quad \exists k.\, (i \leq k < |\pi| \ \wedge \\ \wedge \ \forall j.\, (k \leq j < |\pi| \ \Rightarrow \ (\pi, j) \models \varphi)) \end{aligned}$$

The operator is dual to $\overset{\infty}{\mathsf{F}}$ operator:

$$\begin{aligned} \overset{\infty}{\mathsf{F}}\varphi &\equiv_g \neg\overset{\infty}{\mathsf{G}}\neg\varphi \\ \overset{\infty}{\mathsf{G}}\varphi &\equiv_g \neg\overset{\infty}{\mathsf{F}}\neg\varphi \end{aligned}$$

Past modalities

The definition of past operators is analogous to the definition of future operators. Roughly speaking, a modality has the adjective *past* if its validity for a pointed word (π, i) depends on validity of its immediate subformulae for pointed words (π, j), where $j \leq i$.

$\mathsf{Y}\varphi$ – The past version of X operator called *previously* or *lasttime*. It is sometimes denoted as X^{-1} or $\mathsf{X}_{\exists}^{-1}$ or $\ominus$.

$$(\pi, i) \models \mathsf{Y}\varphi \quad \text{iff} \quad 0 < i \ \wedge \ (\pi, i-1) \models \varphi$$

$\mathsf{Y}_\mathsf{w}\varphi$ – The weak version of Y operator called *weak previously*. It is sometimes denoted as $\tilde{\ominus}$.

$$(\pi, i) \models \mathsf{Y}_\mathsf{w}\varphi \quad \text{iff} \quad 0 < i \ \Rightarrow \ (\pi, i-1) \models \varphi$$

The operator is dual to Y operator:

$$\begin{aligned} \mathsf{Y}\varphi &\equiv_g \neg\mathsf{Y}_\mathsf{w}\neg\varphi \\ \mathsf{Y}_\mathsf{w}\varphi &\equiv_g \neg\mathsf{Y}\neg\varphi \end{aligned}$$

$\varphi \,\mathsf{S}\, \psi$ – The past version of U operator called *since*. It is sometimes denoted as U^{-1}.

$$\begin{aligned} (\pi, i) \models \varphi \,\mathsf{S}\, \psi \quad \text{iff} \quad & \exists k.\, (0 \leq k \leq i \ \wedge \ (\pi, k) \models \psi \ \wedge \\ & \wedge \ \forall j.\, (k < j \leq i \ \Rightarrow \ (\pi, j) \models \varphi)) \end{aligned}$$

$\varphi \,\mathsf{S}_\mathsf{s}\, \psi$ – The past version of U_s called *strict since*. It is sometimes denoted as YS.

$$\begin{aligned} (\pi, i) \models \varphi \,\mathsf{S}_\mathsf{s}\, \psi \quad \text{iff} \quad & \exists k.\, (0 \leq k < i \ \wedge \ (\pi, k) \models \psi \ \wedge \\ & \wedge \ \forall j.(k < j < i \ \Rightarrow \ (\pi, j) \models \varphi)) \end{aligned}$$

Both Y and S operators can be expressed using S_s. Further, S_s can be expressed using the operators Y and S:

$$\begin{aligned} \mathsf{Y}\varphi &\equiv_g \bot \,\mathsf{S}_\mathsf{s}\varphi \\ \varphi \,\mathsf{S}\, \psi &\equiv_g \psi \vee (\varphi \wedge \varphi \,\mathsf{S}_\mathsf{s}\, \psi) \\ \varphi \,\mathsf{S}_\mathsf{s}\, \psi &\equiv_g \mathsf{Y}(\varphi \,\mathsf{S}\, \psi) \end{aligned}$$

$\varphi \,\mathsf{S}_\mathsf{w}\, \psi$ – The past version of U_w operator called *weak since* or *back to*. It is sometimes denoted as B.

$$\begin{aligned} (\pi, i) \models \varphi \,\mathsf{S}_\mathsf{w}\, \psi \quad \text{iff} \quad & \forall j.\, (0 \leq j \leq i \ \Rightarrow \ (\pi, j) \models \varphi) \ \vee \\ & \vee \ \exists k.\, (0 \leq k \leq i \ \wedge \ (\pi, k) \models \psi \ \wedge \\ & \wedge \ \forall j.\, (k < j \leq k \ \Rightarrow \ (\pi, j) \models \varphi)) \end{aligned}$$

The operator is expressively equivalent to S operator:

$$\begin{aligned} \varphi \,\mathsf{S}\, \psi &\equiv_g \varphi \,\mathsf{S_w}\, \psi \wedge \neg(\neg\psi \,\mathsf{S_w}\, \bot) \\ \varphi \,\mathsf{S_w}\, \psi &\equiv_g \varphi \,\mathsf{S}\, \psi \vee \neg(\top \,\mathsf{S}\, \neg\varphi) \end{aligned}$$

$\mathsf{P}\varphi$ – The past version of F operator called *sometime in the past* or *eventually in the past*. It is sometimes denoted as F^{-1} or $\Diamond\!\!\!-$.

$$(\pi, i) \models \mathsf{P}\varphi \quad \text{iff} \quad \exists k.\, (0 \leq k \leq i \;\wedge\; (\pi, k) \models \varphi)$$

The operator can be expressed using S operator:

$$\mathsf{P}\varphi \equiv_g \top \,\mathsf{S}\, \varphi$$

$\mathsf{H}\varphi$ – The past version of G operator called *always in the past* or *hitherto*. It is sometimes denoted as G^{-1} or $\boxminus$.

$$(\pi, i) \models \mathsf{H}\varphi \quad \text{iff} \quad \forall k.\, (0 \leq k \leq i \;\Rightarrow\; (\pi, k) \models \varphi)$$

The operator is dual to P operator:

$$\begin{aligned} \mathsf{P}\varphi &\equiv_g \neg\mathsf{H}\neg\varphi \\ \mathsf{H}\varphi &\equiv_g \neg\mathsf{P}\neg\varphi \end{aligned}$$

$\mathsf{P_s}\varphi$ – The past version of $\mathsf{F_s}$ operator called *sometime in the strict past*. It is sometimes denoted as YP.

$$(\pi, i) \models \mathsf{P_s}\varphi \quad \text{iff} \quad \exists k.\, (0 \leq k < i \;\wedge\; (\pi, k) \models \varphi)$$

The operator can be expressed using the operator $\mathsf{S_s}$ or the operators P and Y. Further, the operator P can be expressed with $\mathsf{P_s}$ operator:

$$\begin{aligned} \mathsf{P_s}\varphi &\equiv_g \top \,\mathsf{S_s}\, \varphi \\ \mathsf{P_s}\varphi &\equiv_g \mathsf{YP}\varphi \\ \mathsf{P}\varphi &\equiv_g \varphi \vee \mathsf{P_s}\varphi \end{aligned}$$

$\mathsf{H_s}\varphi$ – The past version of $\mathsf{G_s}$ operator called *always in the strict past*. It is sometimes denoted as YH.

$$(\pi, i) \models \mathsf{H}\varphi \quad \text{iff} \quad \forall k.\, (0 \leq k < i \;\Rightarrow\; (\pi, k) \models \varphi)$$

The operator is dual to $\mathsf{P_s}$ operator:

$$\begin{aligned} \mathsf{P_s}\varphi &\equiv_g \neg\mathsf{H_s}\neg\varphi \\ \mathsf{H_s}\varphi &\equiv_g \neg\mathsf{P_s}\neg\varphi \end{aligned}$$

$\mathsf{I}\varphi$ – The operator called *initially*. A formula $\mathsf{I}\varphi$ interpreted over a pointed word (π, i) says that φ is valid for the initial position in the word.

$$(\pi, i) \models \mathsf{I}\varphi \quad \text{iff} \quad (\pi, 0) \models \varphi$$

The operator can be seen as a past version of both $\overset{\infty}{\mathsf{F}}$ and $\overset{\infty}{\mathsf{G}}$ operators as each prefix of a word is finite and the formulae $\overset{\infty}{\mathsf{F}}\varphi$ and $\overset{\infty}{\mathsf{G}}\varphi$ interpreted over a finite word say that φ is valid for the last position in the word. The operator can be expressed using P operator:

$$\mathsf{I}\varphi \quad \equiv_g \quad \mathsf{HP}\varphi \quad \equiv_g \quad \neg\mathsf{P}\neg\mathsf{P}\varphi$$

Other modalities

In the following we list the modalities that are not neither future nor past modalities.

$\mathsf{N}\varphi$ – The operator called *now* or *from now on* introduced [LS95]. Roughly speaking, the operator cuts off the history.

$$(\pi, i) \models \mathsf{N}\varphi \quad \text{iff} \quad (\pi_i, 0) \models \varphi$$

It is easy to see that if we restrict the set of temporal operators only to future ones, the addition of N operator is meaningless as $\mathsf{N}\varphi \equiv_g \varphi$ in this case. Further, a formula using only past modalities in combination with N can easily be converted into a globally equivalent formula without N operator in obvious way. Hence, the operator is used in combination with some future modalities and some past modalities as well.

The operator R (and thus also U) can be expressed using the operators N, F, and P. Moreover, the modality $\mathsf{U_s}$ can be expressed using the operators N, F, and $\mathsf{P_s}$:[1]

$$\begin{aligned}
\varphi\,\mathsf{R}\,\psi &\equiv_g \mathsf{NF}(\varphi \wedge \mathsf{H}\psi) \vee \mathsf{G}\psi \\
&\equiv_g \mathsf{NF}(\varphi \wedge \neg\mathsf{P}\neg\psi) \vee \neg\mathsf{F}\neg\psi \\
\varphi\,\mathsf{U_s}\,\psi &\equiv_g \mathsf{NF}(\neg\mathsf{P_s}\top \wedge \psi \wedge \mathsf{H_s}(\varphi \vee \neg\mathsf{P_s}\top)) \\
&\equiv_g \mathsf{NF}(\neg\mathsf{P_s}\top \wedge \psi \wedge \neg\mathsf{P_s}(\neg\varphi \wedge \mathsf{P_s}\top))
\end{aligned}$$

Let us note that the above relations allow to define an expressively equivalent temporal logic containing only unary modalities.

The last operator covered by this overview is called *chop*. The operator has been introduced in [HKP82] and studied in [RP86]. Other two versions

[1] Let us note that the formula $\neg\mathsf{P_s}\top$ is valid only for the first positions in a word.

of chop have been defined in [MS03] and [Mar03a]. Intuitively, using this modality one can divide a word (corresponding to a run of a system) into a prefix (corresponding to a subrun of that run) and a suffix and formulate their temporal specifications separately. The operator has been considered only in connection with future modalities so far and its semantics has been given just for initial positions (semantics for general pointed words is given only in [Mar03a]). As indicated in [RP86], one has to define a global semantics for every pointed word to make this operator "compatible" with past modalities. We define the global semantics for the three versions of chop modality in the following way.

$\varphi \,\mathsf{C_o}\, \psi$ – The *original* version of *chop* operator as introduced in [HKP82] and studied in [RP86]. Intuitively, $\varphi \,\mathsf{C_o}\, \psi$ is valid for a pointed word if the word can be divided into a prefix and a suffix (beginning with the last letter of the prefix) such that the suffix starts at the current or a future position and it satisfies ψ, and φ is valid for the current position in the prefix.

$$(\pi, i) \models \varphi \,\mathsf{C_o}\, \psi \quad \text{iff} \quad \exists k.\ (i \leq k < |\pi| \ \wedge\ (\pi_k, 0) \models \psi \ \wedge \\ \wedge\ (\pi(0, k+1), i) \models \varphi)$$

We recall that $\pi(0, k+1) = \pi(0)\pi(1)\ldots\pi(k)$ and $\pi_k = \pi(k)\pi(k+1)\ldots$. Please note that if a formula $\varphi \,\mathsf{C_o}\, \psi$ is interpreted over an infinite word then φ is interpreted over a finite prefix while ψ is interpreted over an infinite suffix.

$\varphi \,\mathsf{C}\, \psi$ – The version of *chop* operator introduced in [Mar03a] and denoted there as $\diamond$. The difference between this version and the original one is that the prefix and the suffix do not overlap and φ has to be valid for the prefix only on the assumption that the prefix contains the current position.

$$(\pi, i) \models \varphi \,\mathsf{C}\, \psi \quad \text{iff} \quad \exists k.\ (i \leq k < |\pi| \ \wedge\ (\pi_k, 0) \models \psi \ \wedge \\ \wedge\ (i < k \ \Rightarrow\ (\pi(0, k), i) \models \varphi))$$

The operator N can be expressed using the operator C:

$$\mathsf{N}\varphi \quad \equiv_g \quad \bot \,\mathsf{C}\, \varphi$$

$\varphi \,\mathsf{C_s}\, \psi$ – The version of chop operator considered in [MS03] and called *strict chop* here. In contrast to the previous version of chop, the suffix satisfying ψ has to start at a (strictly) future position. Hence, the prefix has to satisfy φ in all cases.

$$(\pi, i) \models \varphi \,\mathsf{C_s}\, \psi \quad \text{iff} \quad \exists k.\ (i < k < |\pi| \ \wedge\ (\pi_k, 0) \models \psi \ \wedge \\ \wedge\ (\pi(0, k), i) \models \varphi)$$

The operator C can be expressed using the operators $\mathsf{C_s}$ and N. Moreover, if we work with future modalities only (as mentioned before, initial and global equivalences coincide in this case) then the operator C can be expressed by using the operator $\mathsf{C_s}$ only:

$$\begin{aligned} \varphi\,\mathsf{C}\,\psi &\equiv_g \varphi\,\mathsf{C_s}\,\psi \vee \mathsf{N}\psi \\ \varphi\,\mathsf{C}\,\psi &\equiv_i \varphi\,\mathsf{C_s}\,\psi \vee \psi \end{aligned}$$

The results about the chop modality cited in this thesis are valid for all three versions of chop. In order to improve the presentation we refer to the operator C only.

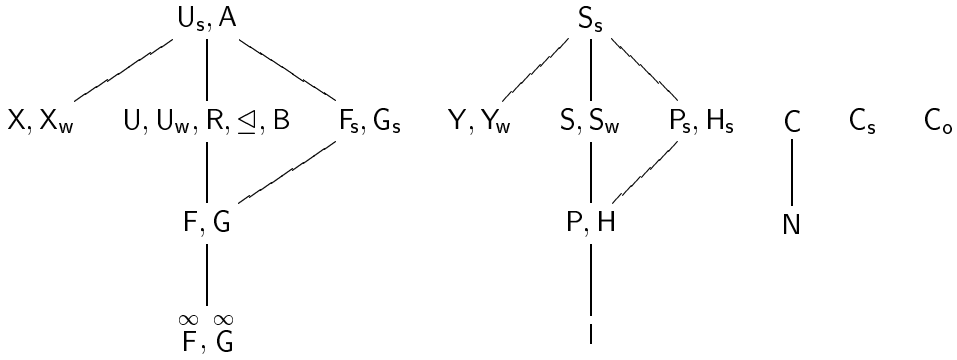

Figure 2.1: Relations between temporal operators.

To sum up, we have introduced 15 future modalities, 10 past modalities, and 4 modalities that are neither past nor future. In many cases we have shown that a modality can be equivalently (with respect to global equivalence) replaced by a formula using another modality and boolean operators. If this relation is symmetric we say that the two operators are expressively equivalent. Hence, the operators can be divided into groups of mutually equivalent operators. Figure 2.1 provides a summary of presented relations between modalities. A line between two groups of equivalent modalities means that each modality from the lower group can be expressed using arbitrary modality from the upper group.

The hierarchies of future and past temporal operators presented in Figure 2.1 are strict. More precisely, modalities from different groups are not expressively equivalent and if two groups are depicted to be incomparable then they are really incomparable (meaning that there exists a formula using a modality from one group such that there is no globally equivalent formula using modalities from the other group instead of the modality, and vice versa). We do not prove our statements about the strictness of the hierarchies here as it is easy to find examples of formulae and pointed words proving them.

The relations depicted in Figure 2.1 often allow to restrict our attention to set of modalities containing one representative of each group. In this case we prefer to work with modalities listed on the first positions in the groups ($\mathsf{U_s}, \mathsf{X}, \mathsf{U}, \mathsf{F_s}, \mathsf{F}, \ldots$). Different situation arises when we put some restrictions on the applicability of boolean operators (e.g. the until-release hierarchy given in Definition 2.10 employs a restriction on the applicability of a negation). In this case, modalities in a group cannot be seen as interchangeable in general.

2.1.4 Fragments

Fragments of a temporal logic are sets of formulae defined by various restrictions on syntax of the logic. A language L is said to be *expressible* or *definable* by/in a fragment $\mathcal{F}$ (or $\mathcal{F}$ *language* for short) if the fragment contains a formula φ such that $L = L^{\Sigma}(\varphi)$ or $L = L^{\Sigma}_{F}(\varphi)$ for some alphabet Σ. We sometimes say 'a fragment' meaning a set of languages expressible by the fragment. Finally, we say that a formula φ is *expressible* in a fragment $\mathcal{F}$ if φ is equivalent to some formula of $\mathcal{F}$.

This section presents all types of fragments we are working with in the following chapters of this thesis. At first we give a general definition covering most of the fragments considered in the literature and then we define the fragments that do not fit into the general scheme.

The general definition employs the notion of nesting depth of a set of modalities.

Definition 2.5 *Let φ be an a formula and M be a set of temporal operators. The* nesting depth *of the operators of M in the formula φ, written M-depth(φ), is defined by induction on the structure of φ. The operators Z and Z' range over unary and binary (temporal as well as boolean) operators respectively.*

$$
\begin{aligned}
M\text{-depth}(\top) &= 0 \\
M\text{-depth}(p) &= 0 \\
M\text{-depth}(Z\varphi) &= \begin{cases} M\text{-depth}(\varphi) + 1 & \text{if } Z \in M \\ M\text{-depth}(\varphi) & \text{otherwise} \end{cases} \\
M\text{-depth}(\varphi_1 Z' \varphi_2) &= \begin{cases} \max\{M\text{-depth}(\varphi_1), M\text{-depth}(\varphi_2)\} + 1 & \text{if } Z' \in M \\ \max\{M\text{-depth}(\varphi_1), M\text{-depth}(\varphi_2)\} & \text{otherwise} \end{cases}
\end{aligned}
$$

To improve our notation we omit the curly brackets whenever M is a singleton.

Example 2.6 *Let $\varphi = p \,\mathsf{U}\, q \vee q \,\mathsf{U}\, \mathsf{X}(p \,\mathsf{U}\, q) \vee \neg\mathsf{X}q$. Then $\{\mathsf{U}, \mathsf{X}\}$-depth$(\varphi) = 3$, U-depth$(\varphi) = 2$, and X-depth$(\varphi) = 1$.*

Definition 2.7 *Let $M_1, M_2, \ldots, M_k$ be mutually disjoint sets of temporal modalities and $m_1, m_2, \ldots, m_k \in \mathbb{N}_0 \cup \{\infty\}$. By $\mathrm{LTL}(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ we denote the set*

$$\{\varphi \mid \varphi \text{ is a formula using only modalities of } M_1 \cup M_2 \cup \ldots \cup M_k \text{ and satisfying } M_i\text{-depth}(\varphi) \leq m_i \text{ for every } 1 \leq i \leq k\}.$$

The set is called simple *fragment if $m_i = \infty$ for all i, and it is called* nesting *fragment otherwise.*

Let $n \in \mathbb{N}_0$. By $\mathrm{LTL}_n(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ we denote the set of all formulae $\varphi \in \mathrm{LTL}(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ such that $At(\varphi) \leq n$ (i.e. the number of atomic propositions occurring in φ is at most n). Further, by $\mathrm{LTL}^+(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ we denote a positive *fragment containing the formulae of $\mathrm{LTL}(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ where no temporal operator is in scope of any negation. By $\mathrm{LTL}^s(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ we denote a* stratified *fragment containing the formulae of $\mathrm{LTL}(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ where no future modality is in scope of any past modality [MP91]. Finally, by $\mathrm{LTL}^{+s}(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k})$ we denote the set*

$$\mathrm{LTL}^+(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k}) \cap \mathrm{LTL}^s(M_1^{m_1}, M_2^{m_2}, \ldots, M_k^{m_k}).$$

To make the notation more readable, we omit the superscripts m_i equal to ∞ and curly brackets delimiting sets M_i with a single element.

For example, $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X})$ denotes nesting fragment of LTL formulae built with temporal operators U and X such that their U-*depth* is at most 3. Further, $\mathrm{LTL}(\mathsf{F}, \mathsf{P})$ is simple fragment of the formulae using modalities F and P. The set of the formulae without any temporal operators can be denoted by $\mathrm{LTL}()$.

The general definition covers also some recognized fragments. For example, $\mathrm{LTL}(\mathsf{X}, \mathsf{F})$ is known as *restricted temporal logic* [PP04], fragments $\mathrm{LTL}(\mathsf{X}, \mathsf{F}, \mathsf{U}^n)$ form the *until hierarchy* [EW00, TW01], and the fragment $\mathrm{LTL}(\mathsf{F}, \mathsf{U})$ called *next-free LTL* is popular in the model checking area. Furthermore, fragments $\mathrm{LTL}(\{\mathsf{U}, \mathsf{S}\}^m, \mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$ form so-called *until-since hierarchy* [TW02]. To shorten our notation, we write US_m instead of $\mathrm{LTL}(\{\mathsf{U}, \mathsf{S}\}^m, \mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$.

Let us note that a formula of a fragment with temporal operators in $M_1 \cup M_2 \cup \ldots \cup M_k$ can contain modalities that are expressible with use of the modalities in the union. The modalities outside the union are formally viewed as abbreviations. For example, $\varphi = a \,\mathsf{U}\, \mathsf{G}b$ can be seen as a formula of fragment $\mathrm{LTL}(\mathsf{U})$ as $\mathsf{G}b$ is an abbreviation for $\neg(\top \,\mathsf{U}\, \neg b)$. In this case, U-*depth*(φ) equals 2. For similar reason, fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ should be seen as LTL fragment containing all future modalities. By analogy, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y})$ denotes the fragment with all future and past modalities (this fragment is sometimes denoted by *LTL+Past*).

In the following we define other fragments of LTL which are studied in this thesis.

Hierarchy of temporal properties

The *hierarchy of temporal properties* has been introduced by Manna and Pnueli [MP90b, CMP92]. The classes of the hierarchy are characterized through four views. Besides language-theoretic, automata, and topological views they have been also defined with use of LTL. The hierarchy contains classes of *safety* properties, *guarantee* properties, *obligation* properties, *response* properties[2], *persistence* properties, and *reactivity* properties. We define two fragments of LTL formulae for each of these classes.

First we define *canonical* fragments.

Definition 2.8 *An LTL formula φ is*

$$\begin{aligned}
\textit{a canonical safety formula} \quad &\textit{iff} \quad \varphi = \mathsf{G}\psi, \\
\textit{a canonical guarantee formula} \quad &\textit{iff} \quad \varphi = \mathsf{F}\psi, \\
\textit{a canonical obligation formula} \quad &\textit{iff} \quad \varphi = \textstyle\bigwedge_{i=1}^{m}(\mathsf{G}\psi_i \vee \mathsf{F}\rho_i), \\
\textit{a canonical response formula} \quad &\textit{iff} \quad \varphi = \overset{\infty}{\mathsf{F}}\psi, \\
\textit{a canonical persistence formula} \quad &\textit{iff} \quad \varphi = \overset{\infty}{\mathsf{G}}\psi, \\
\textit{a canonical reactivity formula} \quad &\textit{iff} \quad \varphi = \textstyle\bigwedge_{i=1}^{m}(\overset{\infty}{\mathsf{G}}\psi_i \vee \overset{\infty}{\mathsf{F}}\rho_i),
\end{aligned}$$

where $m \in \mathbb{N}_0$ and $\psi, \psi_1, \ldots, \psi_m, \rho_1, \ldots, \rho_m$ are formulae of $\mathrm{LTL}(\mathsf{Y}, \mathsf{S})$.

In contrast to the canonical fragments, corresponding *future* fragments [ČP03] are built only with future modalities.

Definition 2.9 *The fragments of* future safety *(φ_S),* future guarantee *(φ_G),* future obligation *(φ_O),* future response *(φ_R), and* future persistence *(φ_P) formulae are defined inductively.*

$$\begin{aligned}
\varphi_S &::= p \mid \neg p \mid \varphi_S \vee \varphi_S \mid \varphi_S \wedge \varphi_S \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S \mid \neg\varphi_G \\
\varphi_G &::= p \mid \neg p \mid \varphi_G \vee \varphi_G \mid \varphi_G \wedge \varphi_G \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G \mid \neg\varphi_S \\
\varphi_O &::= \varphi_S \mid \varphi_G \mid \varphi_O \vee \varphi_O \mid \varphi_O \wedge \varphi_O \mid \mathsf{X}\varphi_O \mid \varphi_O \,\mathsf{U}\, \varphi_G \mid \varphi_O \,\mathsf{R}\, \varphi_S \mid \neg\varphi_O \\
\varphi_R &::= \varphi_S \mid \varphi_G \mid \varphi_R \vee \varphi_R \mid \varphi_R \wedge \varphi_R \mid \mathsf{X}\varphi_R \mid \mathsf{G}\varphi_R \mid \varphi_R \,\mathsf{R}\, \varphi_R \mid \varphi_R \,\mathsf{U}\, \varphi_G \mid \neg\varphi_P \\
\varphi_P &::= \varphi_S \mid \varphi_G \mid \varphi_P \vee \varphi_P \mid \varphi_P \wedge \varphi_P \mid \mathsf{X}\varphi_P \mid \mathsf{F}\varphi_P \mid \varphi_P \,\mathsf{U}\, \varphi_P \mid \varphi_P \,\mathsf{R}\, \varphi_S \mid \neg\varphi_R
\end{aligned}$$

Further, every $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ *formula is a* future reactivity *formula.*

In fact, these future fragments coincide with *standard* fragments [CMP92] without the formulae containing past modalities.

[2] The response properties are sometimes called *recurrence* properties.

Until-release hierarchy

The *until-release hierarchy* of LTL fragments has been introduced in [ČP03]. It is based on the alternation depth of the operators U and R. Therefore it is also called *alternating hierarchy*. The hierarchy consists of fragments of two types, namely UR_i and RU_i.

Definition 2.10 *The fragments* $\mathsf{UR}_i, \mathsf{RU}_i$ *are defined inductively.*

- $\mathsf{UR}_0 = \mathsf{RU}_0 = \mathrm{LTL}(\mathsf{X})$.
- *The fragments* UR_{i+1} *is the least set containing* RU_i *and closed under the application of operators* $\wedge, \vee, \mathsf{X}$, *and* U.
- *The fragment* RU_{i+1} *is the least set containing* UR_i *and closed under the application of operators* $\wedge, \vee, \mathsf{X}$, *and* R.

Let us note that there is no temporal operator in scope of any negation in these fragments.

Deterministic fragment

The *deterministic fragment of LTL*, denoted detLTL, has been identified as the common fragment of LTL and CTL [Mai00].

Definition 2.11 *The fragment* detLTL *is the smallest set of formulae such that*

- *all atomic propositions are in* detLTL,
- *if* φ, ψ *are formulae in* detLTL *and* $p \in At$, *then* $\varphi \wedge \psi$, $\mathsf{X}\varphi$, $(p \wedge \varphi) \vee (\neg p \wedge \psi)$, $(p \wedge \varphi)\,\mathsf{U}\,(\neg p \wedge \psi)$, *and* $\neg((\neg p \vee \neg\varphi)\,\mathsf{U}\,(p \vee \neg\psi))$ *are in* detLTL.

Flat fragment

The idea of *flat fragments* of temporal logics have been introduced in [Dam99] where linear as well as branching temporal logics are under consideration. Although the paper presents only one flat fragment of LTL, namely $\mathrm{flatLTL}(\mathsf{U})$, we give a general definition of flat LTL fragments.

Definition 2.12 *Let* $\mathcal{F}$ *be an LTL fragment. Fragment* $\mathrm{flat}\mathcal{F}$ *consists of all formulae of* $\mathcal{F}$ *such that left subformula of each* U *operator is from* $\mathrm{LTL}()$.

These flat fragments can be alternatively defined with use of the special modality called *flat until* and denoted U^- [DS02, Sch03]. The modality is the until operator where only $\mathrm{LTL}()$ formulae are allowed on its left-hand side.

2.2 First-order monadic logic of order

We recall the syntax, semantics, and definitions of important fragments of *First-Order Monadic Logic of Order (FOMLO)*.

The signature of FOMLO contains unary predicates $P_0, P_1, \ldots$ corresponding to atomic propositions $p_0, p_1, \ldots$ (here we assume that all atomic propositions are of this form) and binary predicates suc and $<$ standing for *successor* and *less than* respectively. We use standard notations for variables, true, boolean connectives, equality, and quantifiers.

The formulae of FOMLO are defined as follows.

- *Atomic formulae* $\top, P_i(x), suc(x,y), x < y$, and $x = y$, where $i \in \mathbb{N}_0$ and x, y are variables, are formulae of FOMLO.
- Let φ, ψ be FOMLO formulae and x be a variable. Then $\neg\varphi, \varphi \wedge \psi, \exists x.\varphi$ are FOMLO formulae as well.

The length of a FOMLO formula φ is denoted by $|\varphi|$. The syntax is further extended with $\bot$, disjunction, and universal quantification defined via standard abbreviations.

Formulae of FOMLO are interpreted over (finite or infinite) words. Let π be a word over an alphabet 2^P, where $P \subseteq At$. With π we associate the sets of natural numbers $M^\pi = \{n \mid 0 \leq n < |\pi|\}$ and

$$P_i^\pi = \{m \mid m \in M^\pi \text{ and } p_i \in \pi(m)\}$$

for every i. We define when a FOMLO formula φ is *valid* for π under a variable assignment v mapping variables to M^π, written $\pi \models \varphi[v]$, by induction on the structure of φ.

$\pi \models \top[v]$

$\pi \models P_i(x)[v]$ iff $v(x) \in P_i^\pi$

$\pi \models suc(x,y)[v]$ iff $v(x) + 1 = v(y)$

$\pi \models x < y[v]$ iff $v(x) < v(y)$

$\pi \models x = y[v]$ iff $v(x) = v(y)$

$\pi \models (\neg\varphi)[v]$ iff $\pi \not\models \varphi[v]$

$\pi \models (\varphi \wedge \psi)[v]$ iff $\pi \models \varphi[v]$ and $\pi \models \psi[v]$

$\pi \models (\exists x.\varphi)[v]$ iff there exists an assignment v' such that $\pi \models \varphi[v']$ and $v'(y) = v(y)$ for every variable y different from x

By $\varphi(x)$ we denote the formula φ where at most the variable x is free. To bring this notation closer to the one for LTL, we write $(\pi, i) \models \varphi$ if $\varphi(x)$ is

valid for a word π under an assignment mapping x to i. Further, we write $\pi \models \varphi$ if $(\pi, 0) \models \varphi$.

Given an alphabet $\Sigma = 2^P$, where P is a finite set of atomic propositions, a FOMLO formula $\varphi(x)$ defines the languages of finite or infinite words over Σ in the following way:

$$L_F^\Sigma(\varphi) = \{u \in \Sigma^+ \mid u \models \varphi\}$$
$$L^\Sigma(\varphi) = \{\alpha \in \Sigma^\omega \mid \alpha \models \varphi\}$$

2.2.1 Fragments

The FOMLO fragments we are dealing with are given by bounds on numbers of variables used in a formula and/or by disallowing the use of suc predicate.

Definition 2.13 *For every $n \in \mathbb{N}$, the fragment FO^n consists of all FOMLO formulae of the form $\varphi(x)$ containing at most n variables. Moreover, by $\mathsf{FO}^n[<]$ we denote the set of formulae from FO^n that contain no suc predicate.*

In the following, we use mainly the fragments FO^3, FO^2, and $\mathsf{FO}^2[<]$.

2.3 Automata and formal languages

This subsection recalls some definitions from the area of automata and formal languages employed in this thesis. We start with basic theory about languages of finite words, then we move to ω-languages, and we close with alternating automata.

2.3.1 Finite words

Definition 2.14 *A* finite automaton *is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite input* alphabet, *Q is a finite set of* states, *$q_0 \in Q$ is an* initial state, *$\delta : Q \times \Sigma \to 2^Q$ is a* transition function, *and $F \subseteq Q$ is a set of* accepting states. *An automaton is called* deterministic *if for all states $q \in Q$ and all $a \in \Sigma$ the set $\delta(q, a)$ is a singleton. The automaton is called* nondeterministic *otherwise. A* run *of the automaton over a word $u \in \Sigma^*$ is a sequence of states $p_0, p_1, \ldots, p_{|u|} \in Q$ such that $p_0 = q_0$ and $p_{i+1} \in \delta(p_i, u(i))$ for all $0 \leq i < |u|$. The run is* accepting *if $p_{|u|} \in F$. A word is* accepted *by the automaton if there is an accepting run of the automaton over the word. We say that the automaton A* recognizes *a language*

$$L(A) = \{u \in \Sigma^* \mid u \text{ is accepted by } A\}.$$

The transition function δ can be extended into a function $\delta^* : Q \times \Sigma^* \to 2^Q$ defined inductively as

$$\delta^*(q, \varepsilon) = \{q\} \quad \text{and} \quad \delta^*(q, ua) = \bigcup_{q' \in \delta^*(q,u)} \delta(q', a),$$

where $q \in Q$, $u \in \Sigma^*$, and $a \in \Sigma$.

Languages recognized by finite automata are called *regular*. A deterministic finite automaton (DFA) A is called *minimal* if there is no DFA with smaller number of states recognizing the language $L(A)$. The fact that for each regular language L there exists a unique (up to the labelling of states) minimal DFA recognizing the language L is a fundamental result of automata theory. The proof can be found in standard textbooks, e.g. in [HU79].

Definition 2.15 *A* regular expression *over an alphabet Σ is given by the following abstract syntax:*

$$R \ ::= \ \varepsilon \mid \emptyset \mid a \mid R \,.\, R \mid \neg R \mid R + R \mid R^*,$$

where a ranges over Σ and operations $.$, $\neg$, $+$, and * denote concatenation, complementation *(with respect to Σ^*),* union, *and* iteration *(also known as* Kleene *or* star closure*) respectively.*

A regular expression R over Σ denotes a language $[R] \subseteq \Sigma^*$ defined by induction on the structure of R:

$$\begin{array}{ll} [\varepsilon] = \{\varepsilon\} & [R \,.\, R'] = \{uv \mid u \in [R], v \in [R']\} \\ [\emptyset] = \emptyset & [\neg R] = \Sigma^* \smallsetminus [R] \\ [a] = \{a\} & [R + R'] = [R] \cup [R'] \\ & [R^*] = [R]^* \end{array}$$

A regular expression without any occurrence of * is called star-free*.*

In the following a regular expression is identified with the language it defines. Regular expressions define exactly regular languages. Languages definable by star-free regular expressions are called *star-free*.

Definition 2.16 *Let $L \subseteq \Sigma^*$ be a language. A* syntactic congruence *of L is an equivalence $\sim_L \subseteq \Sigma^* \times \Sigma^*$ defined as*

$$u \sim_L v \quad \textit{if and only if} \quad \forall w, x \in \Sigma^*.\ wux \in L \Leftrightarrow wvx \in L.$$

Syntactic monoid *of L is the quotient $\Sigma^*|_{\sim_L}$ with the operation concatenation and the class containing ε as an identity element.*

A language is regular if and only if its syntactic monoind is finite.

2.3.2 Infinite words

In contrast to automata over finite words defined above, many various acceptance conditions has been considered in connection with infinite words. In the following we use seven of them.

Definition 2.17 *A finite automaton (over infinite words) is a tuple* $A = (\Sigma, Q, q_0, \delta, F)$*, where* F *is an* acceptance condition*. The meaning of* Σ, Q, q_0*, and* δ *as well as the notions of determinism and nondeterminism are the same as in Definition 2.14. A* run σ *of the automaton over an infinite word* $\alpha \in \Sigma^\omega$ *is a sequence of states* $\sigma = p_0, p_1, \ldots$ *such that* $p_0 = q_0$ *and* $p_{i+1} \in \delta(p_i, \alpha(i))$ *for each* $i \geq 0$*. By* $Inf(\sigma)$ *we denote the set of states occurring infinitely many times in the run* σ*. The set of all states occurring (at least once) in* σ *is denoted by* $Occ(\sigma)$*. The acceptance of a run depends on chosen acceptance condition.*

- Büchi *condition (*$F \subseteq Q$*): a run* σ *is* accepting *iff* $Inf(\sigma) \cap F \neq \emptyset$*.*
- co-Büchi *condition (*$F \subseteq Q$*): a run* σ *is* accepting *iff* $Inf(\sigma) \cap F = \emptyset$*.*
- Streett *condition (*$F = \{(G_1, R_1), \ldots, (G_n, R_n)\}$*, where* $G_i, R_i \subseteq Q$*): a run* σ *is* accepting *iff* $\forall i : (Inf(\sigma) \cap G_i \neq \emptyset) \Rightarrow (Inf(\sigma) \cap R_i \neq \emptyset)$*.*
- Muller *condition (*$F \subseteq 2^Q$*): a run* σ *is* accepting *iff* $Inf(\sigma) \in F$*.*
- occurrence Büchi *condition (*$F \subseteq Q$*): a run* σ *is* accepting *iff* $Occ(\sigma) \cap F \neq \emptyset$*.*
- occurrence co-Büchi *condition (*$F \subseteq Q$*): a run* σ *is* accepting *iff* $Occ(\sigma) \cap F = \emptyset$*.*
- occurrence Streett *condition (*$F = \{(G_1, R_1), \ldots, (G_n, R_n)\}$*, where* $G_i, R_i \subseteq Q$*): a run* σ *is* accepting *iff* $\forall i : (Occ(\sigma) \cap G_i \neq \emptyset) \Rightarrow (Occ(\sigma) \cap R_i \neq \emptyset)$*.*

An ω*-word is* accepted *by the automaton with a given acceptance condition if there is an accepting run of the automaton over the* ω*-word. We say that the automaton* A *with a given acceptance condition* recognizes *a language*

$$L(A) = \{\alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } A\}.$$

The transition function δ can be extended into a function $\delta^* : Q \times \Sigma^* \to 2^Q$ in the same way as in the case of automata over finite words.

We say *Büchi automaton* instead of automaton over infinite words with Büchi acceptance condition. By analogy we use terms *occurrence Büchi automaton, (occurrence) co-Büchi automaton, (occurrence) Street automaton*, and *Muller automaton*.

Languages of ω-words recognizable by nondeterministic Büchi automata are called *ω-regular*. The class of ω-regular languages can equivalently be defined as the class of ω-languages recognized by (deterministic or nondeterministic) Streett or Muller automata.

In this thesis we also employ *weak* and *terminating* nondeterministic Büchi automata.

Definition 2.18 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be a Büchi automaton. The automaton is called* weak *if there exists a partition of the set Q into sets Q_i and an ordering $\leq$ on these sets such that*

- *for each Q_i, either $Q_i \cap F = \emptyset$ or $Q_i \subseteq F$, and*
- *for each $q \in Q_j$, $p \in Q_j$, if $q \in \delta(p, a)$ for some $a \in \Sigma$ then $Q_i \leq Q_j$.*

The automaton is called 1-weak *if it is weak and every Q_i is a singleton. The automaton is called* terminal *if for each $p \in F$ and $a \in \Sigma$ it holds that $\delta(p, a) \neq \emptyset$ and $\delta(p, a) \subseteq F$.*

Several definitions of various syntactic congruences have been proposed for ω-languages (see [MS97]). We present the one introduced by Arnold [Arn85] and sometimes called *iteration syntactic congruence*.

Definition 2.19 *Let $L \subseteq \Sigma^\omega$ be an ω-language. For each word $v \in \Sigma^*$ we define the sets $\Gamma(v)$ and $\Delta(v)$ as*

$$\begin{aligned} \Gamma(v) &= \{(u, w, x) \mid u, w, x \in \Sigma^*, uvwx^\omega \in L\}, \\ \Delta(v) &= \{(u, w, x) \mid u, w, x \in \Sigma^*, u(wvx)^\omega \in L\}. \end{aligned}$$

A syntactic congruence *of L is an equivalence $\sim_L \subseteq \Sigma^* \times \Sigma^*$ defined as*

$$u \sim_L v \quad \textit{if and only if} \quad \Gamma(u) = \Gamma(v) \ \textit{ and } \ \Delta(u) = \Delta(v).$$

Syntactic monoid *of L is a quotient $\Sigma^*|_{\sim_L}$ with the operation concatenation and the class containing ε as an identity element.*

Let us note that the syntactic monoid of ω-regular language is always finite, but there are nonregular ω-languages with finite syntactic monoid.

2.3.3 Alternating automata

In Chapter 7 we work with *alternating automata*. Transition function of an alternating automaton combines the nondeterministic (i.e. existential) and universal mode. More formally, the transition function assigns to each state and letter a positive boolean formula over states. The set of *positive boolean formulae* over a finite set of states Q, denoted $\mathcal{B}^+(Q)$, contains formulae $\top$

(true), $\bot$ (false), all elements of Q and boolean combinations over Q built with $\wedge$ and $\vee$. A subset S of Q is a *model* of $\varphi \in \mathcal{B}^+(Q)$ if the valuation assigning true just to the states in S satisfies φ. A set S is a *minimal model* of φ, denoted $S \models \varphi$, if S is a model of φ and no proper subset of S is a model of φ.

Definition 2.20 *An* alternating Büchi automaton *is a tuple* $A = (\Sigma, Q, q_0, \delta, F)$*, where* Σ, Q, q_0, F *have the same meaning as in nondeterministic Büchi automaton (see Definition 2.17) and* $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ *is a transition function.*

A run of an alternating automaton is a (potentially infinite) tree. A *tree* is a set $T \subseteq \mathbb{N}_0^*$ such that if $xc \in T$, where $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, then also $x \in T$ and $xc' \in T$ for all $0 \leq c' < c$. A *Q-labelled tree* is a pair (T, r) where T is a tree and $r : T \to Q$ is a labelling function.

Definition 2.21 *Let* $A = (\Sigma, Q, q_0, \delta, F)$ *be an alternating Büchi automaton. A* run *of* A *over an* ω*-word* $\alpha \in \Sigma^\omega$ *is a* Q*-labelled tree* (T, r) *with the following properties:*

1. $r(\varepsilon) = q_0$.
2. *For each* $x \in T$ *the set* $S = \{r(xc) \mid xc \in T\}$ *satisfies* $S \models \delta(r(x), w(|x|))$.

A run (T, r) *is* accepting *if for every infinite path* σ *in* T *it holds that* $Inf(\sigma) \cap F \neq \emptyset$*, where* $Inf(\sigma)$ *is a set of all labels (i.e. states) appearing infinitely often on* σ*. An* ω*-word* $\alpha \in \Sigma^\omega$ *is* accepted *by the automaton if there exists an accepting run of* A *over* α*. We say that the alternating Büchi automaton* A recognizes *a language*

$$L(A) = \{\alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } A\}.$$

In Chapter 7 we use the following notation and terminology. Let p be a state of an alternating Büchi automaton $A = (\Sigma, Q, q_0, \delta, F)$. By $A(p)$ we denote the automaton A with initial state p instead of q_0. Further, $Succ(p)$ denotes the set

$$Succ(p) = \{q \mid \exists a \in \Sigma, S \subseteq Q.\ S \models \delta(p, a) \text{ and } q \in S\}$$

of all possible *successors* of p. We also set $Succ'(p) = Succ(p) \setminus \{p\}$. Moreover, we say that a state p has a *loop* whenever $p \in Succ(p)$. Finally, instead of $S \models \delta(a, p)$ we write $p \xrightarrow{a} S$ and say that the automaton has a *transition* leading from p to S under a.

The *alternating 1-weak Büchi automata (A1W automata)* defined below are also known as *very weak alternating Büchi automata* or *linear alternating Büchi automata*.

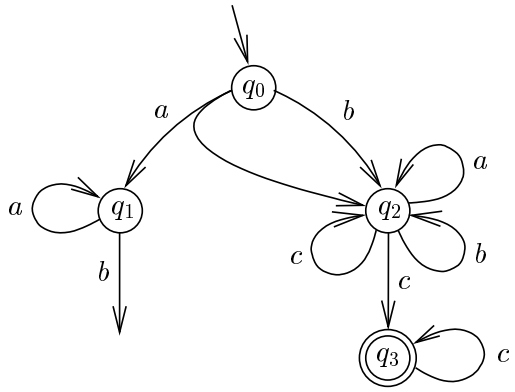

Figure 2.2: The automaton accepting the language $a^*b\{a,b,c\}^*c^\omega$.

Definition 2.22 *An alternating Büchi automaton $A = (\Sigma, Q, q_0, \delta, F)$ is called* 1-weak *if there exists an ordering $<$ on the set Q such that $q \in Succ'(p)$ implies $q < p$.*

An A1W automaton can be depicted as a graph where nodes correspond to the states of the automaton and every transition $p \xrightarrow{a} S$ is represented by a branching edge labelled with a and leading from the node p to the nodes in S. Edges that are not leading to any node correspond to the case when S is an empty set. An initial state is indicated by a special unlabelled edge leading to the corresponding node. The nodes corresponding to the accepting states are double-circled. For example, the Figure 2.2 depicts an automaton accepting the language $a^*b\{a,b,c\}^*c^\omega$.

2.4 Model checking

This section provides a brief introduction into the area of a popular verification method known as *model checking*. In general, the model checking problem is to decide whether a given system satisfies a given specifications. According to the topic of this thesis, we focus on LTL model checking of finite-state systems. In this case, the question is whether all the possible *runs* (or *behaviours*) of a given finite-state system satisfy a given LTL formula (called *specification formula*). We describe the automata-based algorithm proposed in [VW86] and implemented in many model checking tools like, for example, SPIN [Hol97]. For more information about model checking see, for example, [CGP99].

A system to be verified is given in a suitable modelling language (for example, SPIN uses its own C-like language ProMeLa) and it is interpreted as a Kripke structure.

Definition 2.23 *Kripke structure is a tuple* (S, T, s_I, L)*, where*

- S *is a set of* states,
- T *is a set of* transitions *(for each* $t \in T$*,* $t \subseteq S \times S$*),*
- $s_I \in S$ *is an* initial state,
- $L : S \to 2^{At}$ *is a* labelling function *associating to each state a set of atomic propositions that are true in the state.*

We consider deterministic systems only, hence every $t \in T$ *is seen as a partial function* $t : S \to S$*. A* path *in a Kripke structure* K *starting from a state* $s \in S$ *is a maximal (finite or infinite) sequence* $s_0, s_1, \ldots$ *of states such that* $s_0 = s$ *and for every* i *there is a transition* $t_i \in T$ *satisfying* $t_i(s_i) = s_{i+1}$*.*

The validity of an LTL formula for a Kripke structure is defined in the following way.

Definition 2.24 *Let* $K = (S, T, s_I, L)$ *be a Kripke structure. We say that an LTL formula* φ *is* valid *for a state* $s \in S$ *of the Kripke structure* K*, written* $K, s \models \varphi$*, if for every path* $s_0, s_1, \ldots$ *such that* $s_0 = s$ *it holds that* $L(s_0)L(s_1)\ldots \models \varphi$*. Further, we write* $K \models \varphi$ *instead of* $K, s_I \models \varphi$*.*

Formally, the model checking problem is to decide whether a given Kripke structure K and a given specification formula φ satisfy $K \models \varphi$ or not.

For technical reasons, it is assumed that all paths in a Kripke structure are infinite. This can be achieved by adding a special looped transition to every terminal state.

Paths of a Kripke structure represent runs of the system. The labelling function L translates each path $s_0, s_1, \ldots$ into an infinite word $L(s_0)L(s_1)\ldots$ over alphabet 2^{At}. The range of the labelling function is usually restricted to $2^{At(\varphi)}$, where $At(\varphi)$ is a set of atomic propositions occurring in a considered specification formula φ.

Example 2.25 *Figure 2.3 provides an example of a system and associated Kripke structure. The system is a programme beginning with parallel execution of two threads* `A` *and* `B`*. The corresponding Kripke structure reflects the fact that the instructions of* `A` *and* `B` *can be interleaved in arbitrary way. The states carry the information about variables and about the position of control in the two threads. The transitions correspond to individual assignment instructions. In the following, the system is considered in connection with specification formulae with atomic propositions depending on the value of* `x` *only. In Figure 2.3 we therefore explicitly indicate the value of* `x` *in each state of the Kripke structure (the value is considered to be* 0 *at the very beginning). The* $\swarrow$ *direction corresponds to the instructions of* `A`*, and the* $\searrow$ *direction corresponds to the instructions of* `B`*.*

```
cobegin          procedure A()              procedure B()
  A; B;            begin                      begin
coend                for i=1 to 5 do            z = 2;
⋮                      begin                    x = x + 7;
                         x = x + 1;             z = 2 * z;
                         x = x - 1;             z = z - 1;
                       end                    end
                   end
```

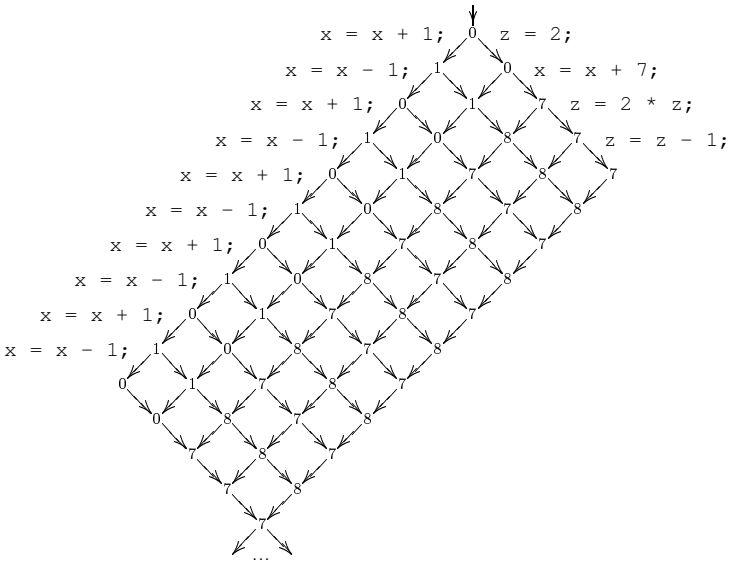

Figure 2.3: Programme and associated Kripke structure.

The classic automata-based model checking algorithm works as follows. The Kripke structure associated with a given system is translated (by moving labels from states to edges) into a Büchi automaton A accepting words corresponding to all possible runs of the system. A given specification formula φ is transformed into a Büchi automaton B such that B recognizes the language of all infinite words (over $2^{At(\varphi)}$) satisfying $\neg\varphi$ (i.e. $L(B) = L(\neg\varphi)$). The automaton B is called the *property automaton*. The original model checking problem to decide whether $L(A) \subseteq L(\varphi)$ is transformed into the problem to decide whether $L(A) \cap L(B) = \emptyset$. Hence, the Büchi automaton C called the *product automaton* and accepting the intersection $L(A) \cap L(B)$ is constructed. It is now sufficient to check if $L(C)$ is empty or not. Algorithms checking the emptiness, including the popular *nested depth-first search*, employ the observation that $L(C)$ is nonempty if and only if the automaton contains a cycle such reachable from the initial state such that there is an accepting state on the cycle. If such a cycle is

found, an infinite word accepted by C can be derived from it. This word corresponds to a run of the system that violates the specification formula.

The running time of the indicated algorithm is linear in the size of a Kripke structure corresponding to the system and exponential in the length of a specification formula. The exponential complexity with respect to the length of a formula is not the main problem as the formula is typically very short. The real problem is that the Kripke structure is usually *very* large comparing to the original description of the system given in a modelling language. This effect, called the *state explosion problem*, has two main reasons, namely the asynchronous parallelism and large data domains. There are various strategies how to deal with this problem. For example, one can reduce the number of states by abstracting the code and/or the data of the system, use various 'compositional' techniques, or use restricted formalisms (like, e.g. pushdown automata) which allow for a kind of 'symbolic' model checking where the explicit construction of the associated Kripke structure is not required. One of the most successful methods is the *partial order reduction* [Val91, God96, HP95, Pel98], also known as the *model checking using representatives*.

2.4.1 Partial order reduction

In fact, the term *partial order reduction* contains several different algorithms based on the same idea: it is not needed to check whether a run of the system satisfies a specification formula if there exists another run that is checked and the two runs cannot be distinguished by the specification formula.

Design of a partial order reduction method has two steps.

1. First, one have to identify a condition (possibly parametrized by a specification formula) such that if a pair of runs satisfies the condition then the runs cannot be distinguished by the specification formula.

2. Second, an algorithm is built. The input of the algorithm is a Kripke structure and a specification formula. The algorithm uses the condition in order to remove some paths from the structure and thus produce a smaller but an equivalent (with respect to the specification formula) Kripke structure.

The condition is usually formulated as an equivalence such that every two equivalent runs are not distinguished by the specification formula. Most partial order reduction methods employ the *stutter equivalence* (see Definition 3.14 or Chapter 5 devoted to stuttering principles). More precisely, the methods uses the fact that two runs cannot be distinguished by any formula φ of $\mathrm{LTL}(\mathsf{U})$ formula φ if the words corresponding to these runs and restricted to alphabet $2^{At(\varphi)}$ are stutter equivalent.

In Chapters 5 and 6 we propose several new equivalences that can be potentially used for the construction of new reduction methods. All of them subsume stutter equivalence. Later we illustrate that applications of these new equivalences can lead to more efficient reductions (meaning that the resulting structures are smaller). Unfortunately, we do not provide any indication of the algorithms based on these equivalences. All the reductions presented are made by hand.

In order to provide a more concrete idea of the partial order reduction algorithms we sketch a basic algorithm for the partial order reduction based on stutter equivalence. This algorithm is implemented, for example, in SPIN and well described in [CGP99].

Let K be a Kripke structure. For all states s of K, by $enabled(s)$ we denote a set of transitions enabled in state s, i.e. $t \in enabled(s)$ if and only if $t(s)$ is defined. For each state s, the reduction algorithm computes a subset of $enabled(s)$ called $ample(s)$. The reduced version of the structure K arises from the original structure by disabling all the transitions leading form a state s that are not in $ample(s)$. Subsequently, some of the states become unreachable in the reduced structure. The sets $ample(s)$ should be large enough as the paths in the reduced structure must contain at least one representative of each class of stutter equivalent paths in the original structure. On the other hand, the reduced structure should be substantially smaller than the original one.

The reduction employs two important principles: *independence* and *invisibility*. An *independence* relation $I \subseteq T \times T$ is a symmetric and antireflexive relation satisfying for every state s and every $(t, t') \in I$ the following condition.

$$\text{If } t, t' \in enabled(s) \text{ then } t \in enabled(t'(s)) \text{ and } t(t'(s)) = t'(t(s)).$$

The *dependency* relation is the complement of the independence relation I.

A transition $t \in T$ is *invisible* with respect to a set of atomic propositions $P \subseteq At$ if for each pair of states s, s' such that $t(s) = s'$ the equation $L(s) \cap P = L(s') \cap P$ holds. In other words, an invisible transition does not change the validity of atomic propositions in P. In the following, invisibility is always associated with a set of atomic propositions occurring in the specification formula.

It has been proved that a reduced system given by sets $ample(s)$ preserves the satisfaction of $\mathrm{LTL}(\mathsf{U})$ specification if for every state s the following four conditions hold.

1. $ample(s) = \emptyset$ iff $enabled(s) = \emptyset$.

2. Along every path in the original structure starting at s it holds that a transition dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occurring first.

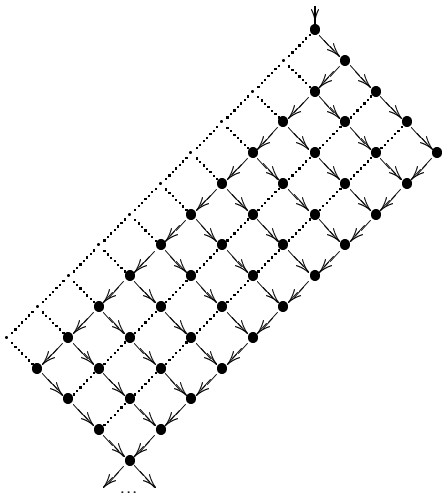

Figure 2.4: The Kripke structure reduced by standard partial order reduction.

3. If $ample(s) \neq enabled(s)$ then every $t \in ample(s)$ is invisible.

4. A cycle is not allowed if it contains a state in which some transition t is enabled, but is never included in $ample(s)$ for any state s on the cycle.

Reduction algorithm uses heuristic for proposing sets $ample(s)$ and it checks if the conditions are satisfied. In the case that for some s every proposed $ample(s)$ violates some condition, the $ample(s)$ is set to $enabled(s)$.

It does not make much sense to construct the whole Kripke structure and then try to reduce it. That is why the reduced system is calculated *on-the-fly*, meaning that a set $ample(s)$ is computed when the model checking algorithm running on the reduced system needs to know the successors of the state s. Hence, for the states that are not reachable in the reduced system the sets $ample(s)$ are not computed at all.

Example 2.26 *Let us consider the system given in Example 2.25 and a specification formula of* $\mathrm{LTL}(\mathsf{U})$ *talking only about value of* x *like, for example,* $\mathsf{G}(\mathrm{x} < 8)$*. The only invisible transitions correspond to the three instructions changing the value of* z*. Using the described partial order reduction method based on the stutter equivalence, the Kripke structure depicted in Figure 2.3 can be reduced into the Kripke structure given in Figure 2.4.*

2.4.2 Model checking a path

Recently, Markey and Schnoebelen [MS03] have introduced a problem of model checking a finite-state system with a single path. The problem is mo-

tivated primarily by runtime verification. A Kripke structure corresponding to a system with a single path has one of the two shapes depicted in Figure 2.5.

Figure 2.5: A Kripke structure with a single finite (left) or infinite (right) path.

Hence, a word corresponding to a path of such system is either *finite* or *ultimately periodic*. Ultimately periodic words are the words of the form uv^ω, where $u, v \in \Sigma^*$ and $v \neq \varepsilon$. They are called *loops* for short.

Chapter 3

Expressiveness

This chapter provides an overview of results about the expressive power of linear temporal logic and its fragments. The expressive power of a formalism (e.g. LTL or its fragment, FOMLO, or Büchi automata) is measured by the languages definable by the formalism.[1] Formally, given fragments $\mathcal{F}_1, \mathcal{F}_2$ of LTL or first-order logic, we write $\mathcal{F}_1 = \mathcal{F}_2$ meaning that every language definable in $\mathcal{F}_1$ is also definable in $\mathcal{F}_2$ and vice versa.

Basically, the results we are going to present can be divided into two groups.

1. Results comparing the expressive power of LTL fragments and the other formalisms mentioned in the previous chapter. These results subsume several translations between the fragments and formalisms under consideration.

2. Results on decidability of the considered LTL fragments. A fragment $\mathcal{F}$ is said to be *decidable* if there exists an algorithm deciding the problem whether a given (regular or ω-regular) language L can be expressed in $\mathcal{F}$ or not. The problem is sometimes called the *membership problem*. If we restrict our attention to languages over strings only, we refer to *decidability (of the membership problem) over strings*. Similarly we talk about *decidability over ω-words*. Complexity of the respective problems is studied as well.

The first group also contains various characterizations of language expressible in the considered fragments. Some characterizations lead directly to the decision algorithms for these fragments. Such characterizations are called *effective*.

Our overview of the expressiveness results is organized as follows. First, we present various characterizations of languages definable in LTL.

[1] Let us note that there is also another natural measure of expressive power, namely the languages of pointed words definable in a given formalism. We do not provide systematic overview of results using this measure. For more details see [Eme90, Wil98].

Most of them are classic results originally presented more then twenty years ago. This part of our overview is partly based on well-written surveys [Eme90] and [Tho90]. The results concerning simple fragments are discussed thereafter. In particular, we give a hierarchy illustrating expressiveness of selected simple fragments. Then we summarize results regarding the other fragments under consideration. The chapter closes with some succinctness issues.

3.1 Complete LTL

First of all we should explain the term *complete LTL*. Originally, we have defined LTL as a temporal logic with two modalities U and X. Later we have added a plethora of other modalities to the logic that is still called LTL. This ambiguity in terminology is justified as these two logics are expressively equivalent, i.e. every language definable in temporal logic with all modalities mentioned in Subsection 2.1.3 is also definable in $\mathrm{LTL}(\mathsf{U},\mathsf{X})$. This is a consequence of the following facts.

Fact 3.1 *Given any formula φ of LTL (with all the mentioned modalities) there exists a globally equivalent formula $\varphi'(x)$ of FOMLO. The construction of $\varphi'(x)$ is given by first-order definition of LTL semantics.*

Theorem 3.2 ([Kam68, GPSS80]) *Given any FOMLO formula $\varphi(x)$ there exists a globally equivalent formula φ' of $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$.*

Theorem 3.3 (Separation Theorem [Gab89]) *Every $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formula φ can be effectively translated into a boolean combination of formulae of $\mathrm{LTL}(\mathsf{U},\mathsf{X}) \cup \mathrm{LTL}(\mathsf{S},\mathsf{Y})$ that is globally equivalent to φ.*

Obviously, any $\mathrm{LTL}(\mathsf{S},\mathsf{Y})$ formula is initially equivalent to a formula without any modality. Hence, every $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formula φ can be effectively translated into a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ that is initially equivalent.

To sum up, every LTL formula with arbitrary modalities is initially equivalent to a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$.

Corollary 3.4 *A language is expressible in LTL if and only if it is expressible in $\mathrm{LTL}(\mathsf{U},\mathsf{X})$.*

3.1.1 Finite words

First, we summarize characterizations of LTL languages over finite words. Roughly speaking, it is known that LTL languages "cannot count modulo n for any $n > 1$". The following definitions formalize this non-counting property in terms of monoids, formal languages, and finite automata.

Definition 3.5 *A finite monoid $(M, \cdot, 1)$ is called* aperiodic *if there exists $n \in \mathbb{N}_0$ such that $x^n = x^{n+1}$ holds for all $x \in M$.*

Definition 3.6 *A language $L \subseteq \Sigma^*$ is* noncounting *if there exists $k \in \mathbb{N}_0$ such that for all $n \geq k$ and $u, v, w \in \Sigma^*$ it holds that*

$$uv^n w \in L \iff uv^{n+1} w \in L.$$

Definition 3.7 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be a finite automaton. A sequence $p_1, \ldots, p_m \in Q$ of distinct states is a* counter *(of length m) for a string $u \in \Sigma^+$ if $m > 1$ and $p_{i+1} \in \delta^*(p_i, u)$ for all $1 \leq i \leq m$, where $p_{m+1} = p_1$. An automaton is* counter-free *if it does not have any counter.*

Theorem 3.8 *Let Σ be an alphabet and $L \subseteq \Sigma^+$ be a language. The following statements are equivalent:*

1. *L is definable in LTL.*
2. *L is definable in FOMLO.*
3. *L is definable in FO^3.*
4. *The minimal deterministic finite automaton recognizing L is counter-free.*
5. *L is definable by a star-free regular expression.*
6. *The syntactic monoid of L is finite and aperiodic.*
7. *L is regular and noncounting.*
8. *L is regular and (m, n)-stutter closed*[2] *for some $m, n \in \mathbb{N}_0$.*

The equivalence of conditions 1 and 2 follows from Fact 3.1 and Theorem 3.2. Further, the equivalence of conditions 1 and 3 follows from Corollary 3.4 and the fact that every formula of $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ can be transformed (using the semantics of U and X given by first-order formulae) into a formula of FO^3. The equivalence of conditions 2, 4, and 5 has been proved by McNaughton and Papert [MP71]. A short proof of equivalence between conditions 2 and 5 can be found in [PP86]. Conditions 5 and 6 are equivalent due to classical theorem of Schützenberger [Sch65]. Finally, it is easy to see that the remaining conditions 7 and 8 are equivalent to condition 4.

Direct translations between LTL formulae and star-free regular expressions can be found in [Zuc86]. The construction of LTL formula accepting the same language as counter-free deterministic finite automaton (DFA) is presented in [Wil99]. Alternative direct proofs of the fact that a counter-free DFA recognizes an LTL language can be found in [CPP93] and [MP90a].

[2] Definition of (m, n)-stutter closed languages is given in Chapter 5.

Some of the conditions presented in Theorem 3.8 lead to algorithms deciding the membership problem for LTL over strings. Namely, given a regular language L one can decide whether its syntactic monoid is aperiodic or whether the corresponding minimal DFA is counter-free. The decision procedure for counter-freeness employs a consequence of a pumping lemma saying that an automaton with n states has a counter if and only if it has a counter for a string u such that $|u| \leq n$. This bound allows to decide the problem whether a DFA is counter-free in polynomial space. Moreover, this problem is PSPACE-complete due to [CH91]. As a minimization of a deterministic finite automaton can be done in PSPACE, we get the following.

Theorem 3.9 *The problem whether a given deterministic finite automaton recognizes an LTL language is* PSPACE*-complete.*

3.1.2 Infinite words

We now move to LTL languages of infinite words where the situation is a bit different. As in the previous case, we start with some definitions.

Definition 3.10 *An ω-language $L \subseteq \Sigma^\omega$ is* noncounting *if there is $k \in \mathbb{N}_0$ such that for all $n \geq k$ and $x, y, z, u \in \Sigma^*$ the two following equivalences hold:*

$$\begin{aligned} xu^n yz^\omega \in L \quad &\Longleftrightarrow \quad xu^{n+1} yz^\omega \in L \\ x(yu^n z)^\omega \in L \quad &\Longleftrightarrow \quad x(yu^{n+1} z)^\omega \in L \end{aligned}$$

Counter-free finite automata over infinite words are defined in the same way as counter-free finite automata over finite words – see Definition 3.7.

The characterizations of LTL ω-languages use the notion of regular expressions in two different ways. The first approach employs an operator which maps languages defined by regular expressions to ω-languages. The characterizations work with two operators of this type.

1. The operator *infinite repetition* (also known as *ω-iteration*). Given a regular expression R, the expression R^ω denotes the language of all ω-words of the form $u_1, u_2, \ldots$, where each $u_i \in R$.

2. The *limit* operator. Given a regular expression R, the expression $\lim R$ denotes the language of all ω-words α such that infinitely many distinct prefixes of α are in R.

The second approach, mentioned in [Eme90], is to embed the regular expressions into LTL. This is done via special operator called *history*, written $[R]_H$, where R is a regular expression. Roughly speaking, $[R]_H$ is true for (π, i) if a word $\pi(0)\pi(1)\ldots\pi(i)$ corresponding to the history of considered computation is in the language defined by R.

$$(\pi, i) \models [R]_H \quad \text{iff} \quad \pi(0, i+1) \in R$$

Theorem 3.11 *Let $L \subseteq \Sigma^\omega$ be an ω-language. The following statements are equivalent:*

1. *L is definable in LTL.*
2. *L is definable in FOMLO.*
3. *L is definable in FO^3.*
4. *$L = \bigcup_{i=1}^m R_i S_i^\omega$, where R_i, S_i are star-free regular expressions such that $S_i.S_i \subseteq S_i$.*
5. *$L = \bigcup_{i=1}^m (\lim R_i \smallsetminus \lim S_i)$ where R_i, S_i are star-free regular expressions.*
6. *L is obtained from Σ^ω by repeated application of boolean operations and concatenation with star-free languages $L' \subseteq \Sigma^*$ on the left.*
7. *$L = \bigcup_{i=1}^m R_i \lim S_i$, where R_i, S_i are star-free regular expressions.*
8. *L is definable by a formula of the form $\bigvee_{i=1}^m (\overset{\infty}{\mathsf{F}}[R_i]_H \wedge \neg \overset{\infty}{\mathsf{F}}[S_i]_H)$, where R_i, S_i are star-free regular expressions.*
9. *L is ω-regular and its syntactic monoid is aperiodic.*
10. *L is ω-regular and noncounting.*
11. *L is recognized by a counter-free Muller automaton.*
12. *L is recognized by an alternating 1-weak Büchi automaton.*
13. *L is ω-regular and (m, n)-stutter closed for some $m, n \in \mathbb{N}_0$.*

Proof of the equivalence of the first three conditions is the same as in Theorem 3.8. Condition 6 has been formulated by Ladner [Lad77] as a definition of *star-free* ω-languages. The equivalence of conditions 2, 4, 5, and 6 has been established by Thomas [Tho79, Tho81]. A short proof of equivalence between conditions 2 and 6 can be found in [PP86]. Conditions 7 and 8 mentioned in [Eme90] are another variations on conditions 4 and 5. Further, Perrin [Per84] showed that an ω-regular language is star-free if and only if its syntactic monoid is aperiodic. The equivalence of conditions 9 and 10 follows directly from Definition 2.19, Definition 3.10, and finiteness of syntactic monoids of ω-regular languages. Condition 11 is equivalent to definability by $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ according to [LPZ85]. The equivalence between conditions 1 and 12 is a corollary of two following facts. First, given an LTL formula φ and an alphabet Σ, one can construct an A1W automaton accepting language $L^\Sigma(\varphi)$ [MSS88]. Second, the translation of A1W automata into language equivalent LTL formulae has been

developed independently in [Roh97] and [LT00]. Deeper connections between LTL and A1W automata are studied in Chapter 7, where we also present the mentioned translations. Finally, the definition of (m, n)-stutter closed languages as well as the proof of the equivalence between LTL definability and (m, n)-stutter closeness can be found in Chapter 5.

As mentioned in [Tho90], the characterization of star-free ω-languages via aperiodicity of syntactic monoid yields the following decidability result.

Theorem 3.12 *The problem whether an ω-language is definable by LTL is decidable.*

Let us note that the requirement $S_i.S_i \subseteq S_i$ in condition 4 cannot be omitted as noted in [Tho79]. This is documented by the following example.

Example 3.13 *Let $R = \varepsilon$, $S_1 = aa \cup b$, and $S_2 = aa \cup ab$ be star-free regular expressions. The languages $L_1 = RS_1^\omega$ and $L_2 = RS_2^\omega$, are not star-free.*[3]

The language L_1 has been used by Ladner [Lad77] to show that star-free ω-languages form a proper subclass of ω-regular languages. The language L_2 can be defined by formula $\mathsf{G}_2 a$. The modality G_2 has been proposed by Wolper [Wol83] in order to demonstrate that there are ω-languages which cannot be defined in *expressively complete* formalisms (the term used at that time for formalisms with the same expressive power as FOMLO or LTL). Wolper proposed an extension of LTL called *Extended Temporal Logic (ETL)*. ETL can express all ω-regular languages. Complexity issues for (several versions of) ETL has been studied in [SC85, VW94, SV89].

As mentioned above, Büchi automata are able to express the wider class of languages than LTL formalism. In spite of this, the translation of $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formulae into Büchi automata is very important as it plays a substantial role in an automata-based approach to the model checking problem (see Section 2.4). The first translation (designed even for the more general fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y})$) has been described in [WVS83]. A survey of translation algorithms together with a tableau-based translation algorithm implemented in SPIN can be found in [GPVW95]. Improved versions of this algorithm are presented in [DGV99, EH00, SB00]. More efficient translation algorithms introduced in [GO01, Tau03] use alternating 1-weak automata as an intermediate formalism.

3.2 Simple fragments

The study of an expressive power of simple fragments concentrates on the fragments built up with future modalities only and the *symmetric fragments*,

[3] For formal proof we refer to Corollary 5.3. We note that Languages S_1^* and S_2^* represent languages of finite words that are regular but not star-free.

i.e. the fragments containing the past counterparts of all the included future modalities as well.

Before we will give an expressiveness hierarchy covering all these fragments, we recall expressiveness and decidability results about simple fragments. The results are divided into three groups presented in the following three subsections.

3.2.1 Stuttering and its implications

First, we present some results related to the well-known stuttering principle.

Definition 3.14 *Let π be a word. A letter $\pi(i)$ is* redundant *if $i+1 < |\pi|$, $\pi(i) = \pi(i+1)$, and π is either finite or there is $j > i$ such that $\pi(j) \neq \pi(i)$. The* canonical form *of π is the word obtained by deletion of all the redundant letters from π. Two words π, π' are* stutter equivalent *if they have the same canonical form. A language L is called* stutter closed *or* stutter-invariant *if it is closed under stutter equivalence, i.e. for every pair of stutter equivalent words π and π' it holds that $\pi \in L$ iff $\pi' \in L$.*

Theorem 3.15 ([Lam83a]) *Every language (of finite or infinite words) expressible in $\mathrm{LTL}(\mathsf{U})$ is stutter closed.*

Theorem 3.16 ([PW97a, Ete00]) *Every stutter closed LTL language (of finite or infinite words) is expressible in $\mathrm{LTL}(\mathsf{U})$.*

As all the languages definable in $\mathrm{LTL}(\mathsf{U},\mathsf{S})$ are stutter closed, the previous theorem implies that $\mathrm{LTL}(\mathsf{U},\mathsf{S}) = \mathrm{LTL}(\mathsf{U})$. It also says that an $\mathrm{LTL}(\mathsf{U_s})$ language is expressible in $\mathrm{LTL}(\mathsf{U})$ if and only if it is stutter closed. This relation can be generalized in the following way.

Theorem 3.17 ([Wil98, Wil99]) *Let $M \subseteq \{\mathsf{F},\mathsf{P},\mathsf{U},\mathsf{S}\}$ and M' be the set containing exactly the strict versions of the modalities in M. A language (of finite or infinite words) defined by a formula of $\mathrm{LTL}(M')$ is expressible in $\mathrm{LTL}(M)$ if and only if the language is stutter closed.*

Proof of Theorem 3.16 provides a translation of arbitrary $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula φ into an $\mathrm{LTL}(\mathsf{U})$ formula $\tau(\varphi)$ such that φ and $\tau(\varphi)$ are equivalent if and only if the language $L(\varphi)$ is stutter closed. As it is decidable whether two formulae are equivalent or not, it is also decidable whether an LTL formula defines a stutter closed language. Complexity of (a more general problem covering also) this problem has been studied in [PWW98].

Theorem 3.18 ([PWW98, Wil98]) *The problem whether a language defined by a finite nondeterministic automaton, Büchi automaton, or by an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula (interpreted over strings or ω-words) is stutter closed is* PSPACE*-complete.*

The PSPACE upper bound for ω-languages given $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae has been alternatively proven in [Ete00].

As a corollary, we get the decidability of the fragment $\mathrm{LTL}(\mathsf{U})$.

Corollary 3.19 *The problem whether an* $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ *formula defines a language (of finite or infinite words) expressible in* $\mathrm{LTL}(\mathsf{U})$ *is* PSPACE*-complete.*

3.2.2 Forbidden patterns

Several simple fragments can be characterized by means of *forbidden patterns*. The concept of forbidden patterns has been originally introduced in [CPP93] under the name *forbidden configuration*. Roughly speaking, a language L can be defined by a fragment if and only if a minimal DFA recognizing the reverse of L does not have a structural property given by the corresponding forbidden pattern. The method can be used for languages of finite words only.

Given a set N, an *N-labelled digraph* is a tuple (V, E) where V is an arbitrary set of vertices and $E \subseteq V \times N \times V$ is a set of edges. The *transition graph* of a DFA $A = (\Sigma, Q, q_0, \delta, F)$ is the Σ^*-labelled digraph (Q, E) where

$$E = \{(q, u, q') \mid q \in Q, u \in \Sigma^*, \text{ and } \delta^*(q, u) = \{q'\}\}.$$

So the transition graph of any DFA has a finite number of vertices but potentially infinitely many edges.

A *pattern* is a labelled digraph whose vertices are *state variables* and whose edges are labelled with *variables for labels* of two different types: variables ranged over *nonempty* strings (denoted by $u, v, \ldots$) and variables ranged over alphabet symbols (denoted $a, b, \ldots$). In addition, a pattern is occupied by side conditions stating which state variables must be interpreted as distinct states. The patterns are drawn as graphs and the notation of side conditions is such that all vertices marked by "$\bullet$" must be mutually distinct.

We say that Σ^*-labelled digraph *matches a pattern* if there is an assignment to the variables satisfying the type constraints as well as the side conditions so that the digraph obtained by replacing each variable by the value assigned to it is a subgraph of the given digraph.

Let $u = u(0)u(1)\ldots u(n-1)$ be a string. The *reverse* of u is defined as $u^R = u(n-1)u(n-2)\ldots u(1)u(0)$. Similarly, the reverse of language L is defined as $L^R = \{u^R \mid u \in L\}$.

The following collection of forbidden patterns has been presented in [Wil99]. The forbidden patterns characterizing full LTL correspond to counter-freeness as discussed in Section 3.1. The forbidden patterns for $\mathrm{LTL}(\mathsf{X},\mathsf{F})$ and $\mathrm{LTL}(\mathsf{F_s})$ have been deduced in [EW00] with use of Ehrenfeucht-Fraïssé games. The characterization of $\mathrm{LTL}(\mathsf{X},\mathsf{F})$ has been originally introduced in [CPP93], where the authors proposed the name

Restricted Temporal Logic (RTL) for the fragment $\mathrm{LTL}(\mathsf{X},\mathsf{F})$. The other patterns are easy to obtain.

Theorem 3.20 ([MP71, GPSS80, CPP93, EW00, PW97a]) *Let $\mathcal{F}$ be one of the fragments $\mathrm{LTL}()$, $\mathrm{LTL}(\mathsf{X})$, $\mathrm{LTL}(\mathsf{F}_s)$, $\mathrm{LTL}(\mathsf{X},\mathsf{F})$, and LTL. A regular language L is expressible in $\mathcal{F}$ if and only if the transition graph of the minimal DFA recognizing L^R does not match the pattern(s) for $\mathcal{F}$ depicted in Figures 3.1–3.5. Further, a regular language L is stutter closed if and only if the transition graph of the minimal DFA recognizing L^R does not match the pattern depicted in Figure 3.6.*

Due to Theorem 3.17, the fragments $\mathrm{LTL}(\mathsf{F})$ and $\mathrm{LTL}(\mathsf{U})$ are characterized by forbidden pattern depicted in Figure 3.6 together with the pattern(s) corresponding to $\mathrm{LTL}(\mathsf{F}_s)$ or LTL respectively.

These characterizations are effective due to the following theorem. Let us note that the number of patterns that must be tested in the case of full LTL is bounded by the number of states of the minimal DFA under consideration.

Theorem 3.21 ([Wil98]) *Let (V,E) be a pattern. The following problem is in* PSPACE. *Given an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula φ, does a transition graph of a minimal DFA recognizing the reverse of language $L_F(\varphi)$ match the pattern (V,E)?*

Corollary 3.22 *Let $\mathcal{F}$ be one of the fragments $\mathrm{LTL}()$, $\mathrm{LTL}(\mathsf{X})$, $\mathrm{LTL}(\mathsf{F}_s)$, $\mathrm{LTL}(\mathsf{F})$ $\mathrm{LTL}(\mathsf{X},\mathsf{F})$, or $\mathrm{LTL}(\mathsf{U})$. The problem whether a language of strings given by an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula can be defined by $\mathcal{F}$ is* PSPACE*-complete.*

The PSPACE-hardness of these problems has been proven in [Wil98] by reduction of the satisfiability problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae (which is known to be PSPACE-complete [SC85]) to each of the respective problems.

Let us note that characterizations of several LTL fragments in terms of patterns forbidden for minimal DFAs recognizing directly a language L instead of its reverse can be found in [Wil98].

3.2.3 Connection to FOMLO and its implications

The identity of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ and FO^3 has been discussed in Section 3.1 already. Here we recall another two identities proven in [EVW02]. Some decidability results implied by these identities will be mentioned as well.

Theorem 3.23 ([EVW02]) *A language is expressible in FO^2 if and only if it is expressible in $\mathrm{LTL}(\mathsf{F},\mathsf{X},\mathsf{P},\mathsf{Y})$. Further, a language is expressible in $\mathsf{FO}^2[<]$ if and only if it is expressible in $\mathrm{LTL}(\mathsf{F}_s,\mathsf{P}_s)$.*

Later on[4], it has been shown that expressive power of these fragments coincides with another FOMLO fragments.

[4]The paper [EVW02] is a journal version a conference paper presented at LICS 1997.

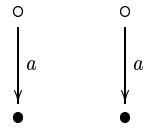

Figure 3.1: Forbidden pattern for $\mathrm{LTL}()$.

Figure 3.2: Forbidden pattern for $\mathrm{LTL}(\mathsf{X})$.

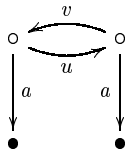

Figure 3.3: Forbidden pattern for $\mathrm{LTL}(\mathsf{F_s})$.

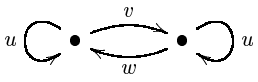

Figure 3.4: Forbidden pattern for $\mathrm{LTL}(\mathsf{X}, \mathsf{F})$.

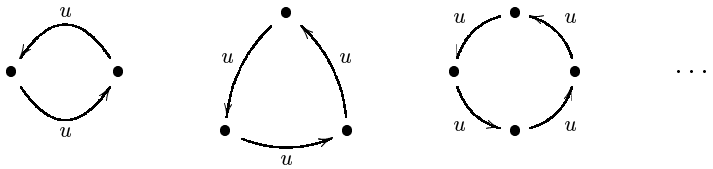

Figure 3.5: Forbidden patterns for LTL.

Figure 3.6: Forbidden pattern for stutter-invariance.

Definition 3.24 *The FOMLO fragment Σ_2 consists of all FOMLO formulae of the form $\varphi(x) = \exists x_0 \ldots \exists x_m \forall y_0 \ldots \forall y_n \psi$, where ψ does not contain any quantifier. Similarly, Π_2 consists of FOMLO formulae of the form $\varphi(x) = \forall x_0 \ldots \forall x_m \exists y_0 \ldots \exists y_n \psi$, where ψ is quantifier-free. By $\Sigma_2 \cap \Pi_2$ we denote a class of languages definable by a Σ_2 formula as well as by a Π_2 formula. By analogy, $(\Sigma_2 \cap \Pi_2)[<]$ contains languages definable by both Σ_2 and Π_2 formulae without predicate suc.*

Theorem 3.25 ([TW98]) *A language of finite words is expressible in $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$ if and only if it is in $\Sigma_2 \cap \Pi_2$. Further, a language of finite words is expressible in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ if and only if it is in $(\Sigma_2 \cap \Pi_2)[<]$.*

Let us note that the theorem cannot be extended to ω-languages.

The combination of the theorem above and the result presented in [PW97b] brings another characterization of $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ languages.

Definition 3.26 *Let Σ be an alphabet. A product $B_0^* a_1 B_1^* a_2 \ldots a_n B_n^*$, where $B_i \subseteq \Sigma$ and $a_i \in \Sigma$, is* unambiguous *if for every $u \in \Sigma^+$ there is at most one sequence $(u_0, u_1, \ldots, u_n)$ such that $u_i \in B_i^*$ and $u = u_0 a_1 u_1 a_2 \ldots a_n u_n$. A language of strings over Σ is* unambiguous *if it is a disjoint union of unambiguous products.*

Theorem 3.27 ([TW98]) *A language of finite words is definable in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ if and only if it is unambiguous.*

Theorem 3.27 provides an important and fruitful connection between LTL fragments and deep algebraic results. The two following theorems are corollaries of this connection. For details and other algebraic characterizations of the fragments $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ and $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$ we refer to [TW98] and [Wil98].

Theorem 3.28 ([TW98, Wil98]) *The fragments $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$ and $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ are decidable (over strings as well as over ω-words).*

Moreover, the following complexity bounds for decidability over strings are known.

Theorem 3.29 ([TW98]) *The problem of determining whether a given $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formula defines a language of strings expressible in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ is in* EXPSPACE*. The problem is* PSPACE*-hard at the same time. These bounds remain unchanged if we replace the fragment $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ with $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$.*

The exact complexity of the problem for $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ and $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$ is an open question formulated in [TW98].

Theorems 3.17, 3.18, and 3.29 imply that the fragment $\mathrm{LTL}(\mathsf{F}, \mathsf{P})$ is decidable in EXPSPACE as well.

3.2.4 Expressiveness hierarchy

Three hierarchies reflecting relative expressive power of LTL fragments have been presented so far. Two of them have been introduced in [Wil98]. The first contains only the fragments built with some of the modalities $\mathsf{X}, \mathsf{F}, \mathsf{F_s}$, and U, while the second consists of symmetric versions of these fragments. The hierarchy presented in [Mar03a] covers the fragments built with the operators $\mathsf{X}, \mathsf{F}, \mathsf{U}$ and symmetric versions of these fragment.

The hierarchy depicted in Figure 3.7 subsumes three hierarchies mentioned above. An edge between two fragments means that the languages expressible in the lower fragment form a strict subclass of the languages expressible in the higher fragment. The hierarchy covers all the fragments built up with some future modalities and symmetric versions of these fragments; every such a fragment is either represented explicitly or it coincides with some of the represented fragments due to the relations between modalities described in Subsection 2.1.3.

The shape of the hierarchy is given by definitions of the considered fragments, statements presented hereinbefore in this section, straightforward observations on expressiveness of the fragments, and the following results.

The following two theorems are formulated and proven in PhD theses of Markey and Laroussinie.

Theorem 3.30 ([Mar03a]) *Fragments* $\mathrm{LTL}(\mathsf{F})$ *and* $\mathrm{LTL}(\mathsf{F}, \mathsf{X})$ *are strictly less expressive than their respective symmetric versions.*

The proof can be also modified to show that the fragment $\mathrm{LTL}(\mathsf{F_s})$ is strictly less expressive than its symmetric version.

Theorem 3.31 ([Lar94]) *There is a language expressible in* $\mathrm{LTL}(\mathsf{U})$ *that is not expressible in* $\mathrm{LTL}(\mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$.

As a corollary of this theorem we get that $\mathrm{LTL}(\mathsf{F}, \mathsf{P})$ is strictly less expressive than $\mathrm{LTL}(\mathsf{U})$.

Finally, we present one original result.

Theorem 3.32 *Fragment* $\mathrm{LTL}(\mathsf{X})$ *is strictly less expressive than* $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$.

Proof: One can easily prove that there are languages definable in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ but not in $\mathrm{LTL}(\mathsf{X})$. We demonstrate that every $\mathrm{LTL}(\mathsf{X})$ language can be defined in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$. By $\mathsf{X}^n\varphi$ we denote a formula

$$\overbrace{\mathsf{X}\mathsf{X}\ldots\mathsf{X}}^{n}\varphi.$$

The expressions $\mathsf{F_s}^n$ and $\mathsf{P_s}^n$ have analogous meanings. It is sufficient to deal with $\mathrm{LTL}(\mathsf{X})$ formulae of the form $\mathsf{X}^n\varphi$ where $n \geq 0$ and φ is a formula without any temporal operator as every $\mathrm{LTL}(\mathsf{X})$ formula can be converted

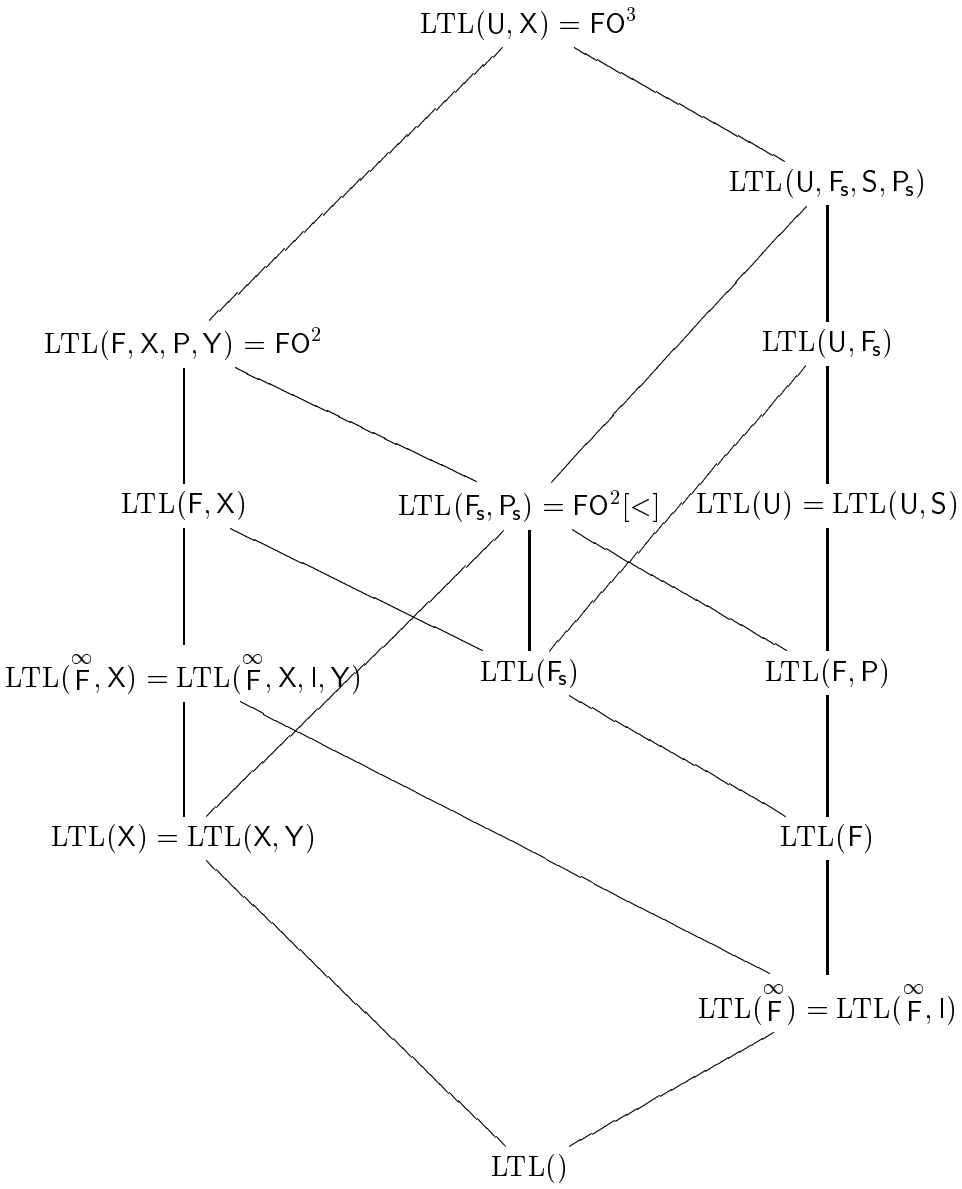

Figure 3.7: Expressiveness hierarchy of simple fragments.

into a globally equivalent boolean combination of these formulae. To finish this proof we demonstrate that the following two formulae are initially equivalent (assuming that $\varphi \in \mathrm{LTL}()$):

$$\mathsf{X}^n\varphi \quad \equiv_i \quad \mathsf{F_s}^n(\varphi \wedge \neg \mathsf{P_s}^{n+1}\top)$$

A word π satisfies $\mathsf{F_s}^n(\varphi \wedge \neg \mathsf{P_s}^{n+1}\top)$ if and only if there exists k such that $n \leq k < |\pi|$ and $(\pi, k) \models \varphi \wedge \neg \mathsf{P_s}^{n+1}\top$. Please observe that formula $\neg \mathsf{P_s}\top$ is valid just for the initial position in an arbitrary word. In general, $(\pi, k) \models \neg \mathsf{P_s}^{n+1}\top$ if and only if $k \leq n$. Hence, $(\pi, k) \models \varphi \wedge \neg \mathsf{P_s}^{n+1}\top$ is equivalent to $(\pi, k) \models \varphi$ and $k \leq n$. To sum up, $\pi \models \mathsf{F_s}^n(\varphi \wedge \neg \mathsf{P_s}^{n+1}\top)$ if and only if $n < |\pi|$ and $(\pi, n) \models \varphi$. We are done as this is exactly the meaning of $\pi \models \mathsf{X}^n\varphi$. ■

The theorem has some interesting corollaries. First, as the fragments $\mathrm{LTL}(\mathsf{X})$ and $\mathrm{LTL}(\mathsf{F_s})$ have incomparable expressive power, $\mathrm{LTL}(\mathsf{F_s})$ is strictly less expressive than $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$. Similarly, as the expressiveness of $\mathrm{LTL}(\mathsf{X})$ and $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s})$ is incomparable and there are languages expressible in $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s})$ but not in $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ (see Theorem 3.31), we get that fragments $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ and $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s})$ are incomparable too. This immediately implies that the fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s}, \mathsf{S}, \mathsf{P_s})$ is strictly more expressive than both $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ and $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s})$.[5]

3.3 Nesting fragments

Only a few kinds of nesting fragments have been studied so far. Researchers focuses on fragments of the form $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$, US_k, $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$. Fragments of each of these kinds form a corresponding hierarchy. Two standard questions studied for all of these hierarchies is whether the hierarchy is *(semantically) strict* and whether it is *decidable*. The precise meaning of the strictness and the decidability will be given soon.

In Section 7.3 we present an automata-based characterization of language classes corresponding to fragments of the form $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$. In fact, this characterization is formulated for all fragments of a more general form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k)$, where $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$.

3.3.1 Fragments $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$: Until hierarchy

Fragments of the form $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$ form the hierarchy

$$\mathrm{LTL}(\mathsf{U}^0, \mathsf{F}, \mathsf{X}) \subseteq \mathrm{LTL}(\mathsf{U}^1, \mathsf{F}, \mathsf{X}) \subseteq \mathrm{LTL}(\mathsf{U}^2, \mathsf{F}, \mathsf{X}) \subseteq \ldots$$

[5] The statement that fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s})$ is not as expressive as its symmetric counterpart $\mathrm{LTL}(\mathsf{U}, \mathsf{F_s}, \mathsf{S}, \mathsf{P_s})$ has been already formulated (without any evidence) in [Wil98].

called *until hierarchy*.

In [EW00] it has been shown that the hierarchy is semantically strict, i.e. for every $k \in \mathbb{N}_0$ the class of languages definable by $\mathrm{LTL}(\mathsf{U}^{k+1}, \mathsf{F}, \mathsf{X})$ is strictly larger than the class of languages definable by $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$. The proof employs a parametrized language FAIR_{k+1} (over a three-letter alphabet) expressible in $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$ but not in $\mathrm{LTL}(\mathsf{U}^{k-1}, \mathsf{F}, \mathsf{X})$ (this is proven with use of an appropriate Ehrenfeucht-Fraïssé game designed in the paper).

The decidability of the until hierarchy has been proven in [TW01] applying deep results from finite semigroup theory.

Theorem 3.33 ([TW01]) *Given a regular or ω-regular language L and $k \in \mathbb{N}_0$, it is decidable whether L is definable by $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$ or not.*

Corollary 3.34 *For every language given by an LTL formula, one can compute the minimal k such that the language is expressible in $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$.*

It is worth mentioning here that the paper [TW01] also deals with decidability of fragments of the form $\mathrm{LTL}(\mathsf{S}^k, \mathsf{P}, \mathsf{Y})$ constituting the *since hierarchy*. However, languages expressed by these fragments are defined in a bit different way: given an alphabet Σ, an $\mathrm{LTL}(\mathsf{S}^k, \mathsf{P}, \mathsf{Y})$ formula φ defines the language

$$L^{\Sigma}(\varphi) = \{u \in \Sigma^+ \mid (u, |u|-1) \models \varphi\}.$$

3.3.2 Fragments US_k: Until-since hierarchy

Let us recall that US_k denotes the fragment $\mathrm{LTL}(\{\mathsf{U}, \mathsf{S}\}^k, \mathsf{F}, \mathsf{X}, \mathsf{P}, \mathsf{Y})$.

The strictness of the until-since hierarchy has been shown in [EW00] with use of Ehrenfeucht-Fraïssé game.

As the fragment US_0 coincides with $\mathrm{LTL}(\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P})$, its decidability has been given by Theorem 3.28 already. The decidability of all US_k fragments over strings have been proven in [TW02] with use of deep algebraic results.

Theorem 3.35 *[TW02] For every $k \geq 0$, the fragment US_k is decidable over strings.*

Authors of the theorem state that they can prove that fragments US_0 and US_1 are decidable also over ω-words and it is possible that their proof technique can be generalized to higher levels of the until-since hierarchy.

3.3.3 Fragments $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F}_\mathsf{s})$

The hierarchy of $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F}_\mathsf{s})$ fragments, also known as the *next hierarchy of restricted temporal logic*, has been studied in [Sch00]. The paper works only with languages over strings.

Two characterizations of $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$ fragments have been formulated and proven. The first one employs the notion of k-rightdeterministic languages while the second one gives a corresponding forbidden pattern.

Definition 3.36 *Let Σ be an alphabet, $k \geq 0$, $n > 0$, $u_0, \ldots, u_n \in \Sigma^{k+1}$, and $B_0, \ldots, B_n \subseteq \Sigma$. By $(u_0 B_0 u_1 B_1 \ldots u_n B_n)_k$ we denote a language of all strings $v \in \Sigma^+$ such that there exists a sequence of numbers $0 = j_0 < j_1 < \ldots < j_n < |v|$ satisfying for every $0 \leq i \leq n$ the following conditions*[6]

- $v(j_i, k+1) = u_i$,
- $v(j, k+1) \in B_i$ *for every* $j_i < j < j_{i+1}$, *where* $j_{n+1} = |v|$.

Further, a language of the form $(u_0 B_0 u_1 B_1 \ldots u_n B_n)_k \cup D$ is called k-rightdeterministic *if $D \subseteq \{w \in \Sigma^+ \mid |w| \leq k\}$ and $u_i \notin B_i$ holds for every $1 \leq i \leq n$.*

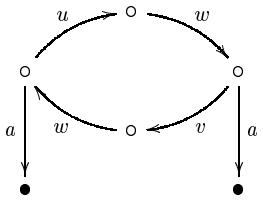

Figure 3.8: Forbidden pattern for $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$ with side condition $|w| = k$. In particular, $w = \varepsilon$ for $k = 0$ (this is an exception to the definition of patterns saying that w ranges over nonempty strings).

Theorem 3.37 ([Sch00]) *Let L be a regular language. The following statements are equivalent:*

1. *L is definable by $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$.*
2. *L is a finite union of k-rightdeterministic languages.*
3. *The transition graph of the minimal DFA recognizing L^R does not match the pattern depicted in Figure 3.8.*

One can observe that the forbidden pattern for $\mathrm{LTL}(\mathsf{X}^0, \mathsf{F_s})$ coincides with the forbidden pattern for $\mathrm{LTL}(\mathsf{F_s})$ (see Figure 3.3). Moreover, it is easy to prove that the transition graph of a DFA does not match the pattern for $\mathrm{LTL}(\mathsf{X}, \mathsf{F})$ (see Figure 3.4) if and only if there exists some $k \in \mathbb{N}_0$ such that the transition graph does not match the pattern for $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$.

The characterization via forbidden patterns together with Theorem 3.21 gives us the decidability of the hierarchy.

[6] We recall that $v(j, k+1)$ denotes the subword $v(j)v(j+1)\ldots v(j+k)$.

Corollary 3.38 *For every $k \geq 0$, the problem whether a language of strings given by an $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formula can be expressed in $\mathrm{LTL}(\mathsf{X}^k, \mathsf{F_s})$ is* PSPACE*-complete.*

As in the case of Corollary 3.22, the PSPACE-hardness can be proven by reduction of the satisfiability problem for $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formulae.

The characterization in terms of forbidden patterns is also applied in the proof of the fact that the next hierarchy of restricted temporal logic is semantically strict.

We note that the paper [Sch00] provides also forbidden patterns for minimal DFAs recognizing directly a language under examination instead of its reverse. Furthermore, analogous results are formulated for fragments $\mathrm{LTL}(\mathsf{Y}^k, \mathsf{P_s})$, where the languages expressed by these fragments are defined in the way described at the end of Subsection 3.3.1.

3.3.4 Fragments $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$

These fragments are studied in Chapters 5, 6, and 7. We here summarize the main results regarding the hierarchies of the fragments under consideration.

The $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$ hierarchies are strict. Further, the hierarchy of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragments is strict in the sense that a fragment $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ is strictly less expressive than each of the fragments $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n+1})$. In fact, we show a bit stronger statement.

Theorem 3.39 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be fragments of the form $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ or $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$ (not necessarily of the same form) such that $\mathcal{F}_1$ is syntactically not included in $\mathcal{F}_2$. Then there is a language expressible in $\mathcal{F}_1$ which cannot be defined in $\mathcal{F}_2$.*

Given a finite alphabet, a fragment of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ defines only finitely many languages. Hence, every such a fragment is decidable. A decidability of $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ fragments is proven with use of n-stuttering (for details see [KS02] dealing with languages of strings or Chapter 5 dealing with ω-languages).

Theorem 3.40 *For every $n \geq 0$, the problem whether a language (of finite or infinite words) given by an $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formula can be expressed in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ is* PSPACE*-complete.*

Further, Chapter 6 provides a characterization of ω-languages expressible by $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ in terms of *characteristic patterns*[7].

[7]Please note that there is no connection to forbidden patterns.

3.4 Other fragments

3.4.1 Hierarchy of temporal properties

This hierarchy has been presented and studied in [MP90b], [CMP92], and [ČP03]. All these papers work with languages of infinite words only.

The characterizations of (canonical or standard) safety, guarantee, obligation, response, persistence, and reactivity fragments are based on the corresponding classes of (not only LTL definable) ω-languages. The definitions of these classes employ four operators constructing ω-languages from languages of strings. Let $L \subseteq \Sigma^+$ be a language of nonempty strings.

- The language $A(L)$ consists of all ω-words α such that every nonempty prefix of α belongs to L.
- The language $E(L)$ consists of all ω-words α such that there exists a prefix of α that belongs to L.
- The language $R(L)$ consists of all ω-words α such that infinitely many prefixes of α belong to L.
- The language $P(L)$ consists of all ω-words α such that all but finitely many prefixes of α belong to L.

Definition 3.41 *An ω-language L is*

$$\begin{array}{rll}
\textit{a} \text{ safety } \textit{property} & \textit{iff} & L = A(L') \textit{ for some } L' \subseteq \Sigma^+, \\
\textit{a} \text{ guarantee } \textit{property} & \textit{iff} & L = E(L') \textit{ for some } L' \subseteq \Sigma^+, \\
\textit{an} \text{ obligation } \textit{property} & \textit{iff} & L = \bigcap_{i=1}^{m} (A(L'_i) \cup E(L''_i)), \\
 & & \textit{where } m > 0 \textit{ and } L'_i, L''_i \subseteq \Sigma^+ \textit{ for every } i, \\
\textit{a} \text{ response } \textit{property} & \textit{iff} & L = R(L') \textit{ for some } L' \subseteq \Sigma^+, \\
\textit{a} \text{ persistence } \textit{property} & \textit{iff} & L = P(L') \textit{ for some } L' \subseteq \Sigma^+, \\
\textit{a} \text{ reactivity } \textit{property} & \textit{iff} & L = \bigcap_{i=1}^{m} (R(L'_i) \cup P(L''_i)), \\
 & & \textit{where } m > 0 \textit{ and } L'_i, L''_i \subseteq \Sigma^+ \textit{ for every } i.
\end{array}$$

The classes of safety, guarantee, obligation, response, persistence, and reactivity properties are closed under intersection and union, the obligation and reactivity classes are also closed under complementation. An ω-language is a safety property if and only if its complement is a guarantee property. The same relation holds for response and persistence properties as well. The inclusions between the classes are indicated in the hierarchy depicted in Figure 3.9, where an edge between two classes means that the lower class is a strict subclass of the upper class. Moreover, the obligation class is exactly the intersection of the response and persistence classes.

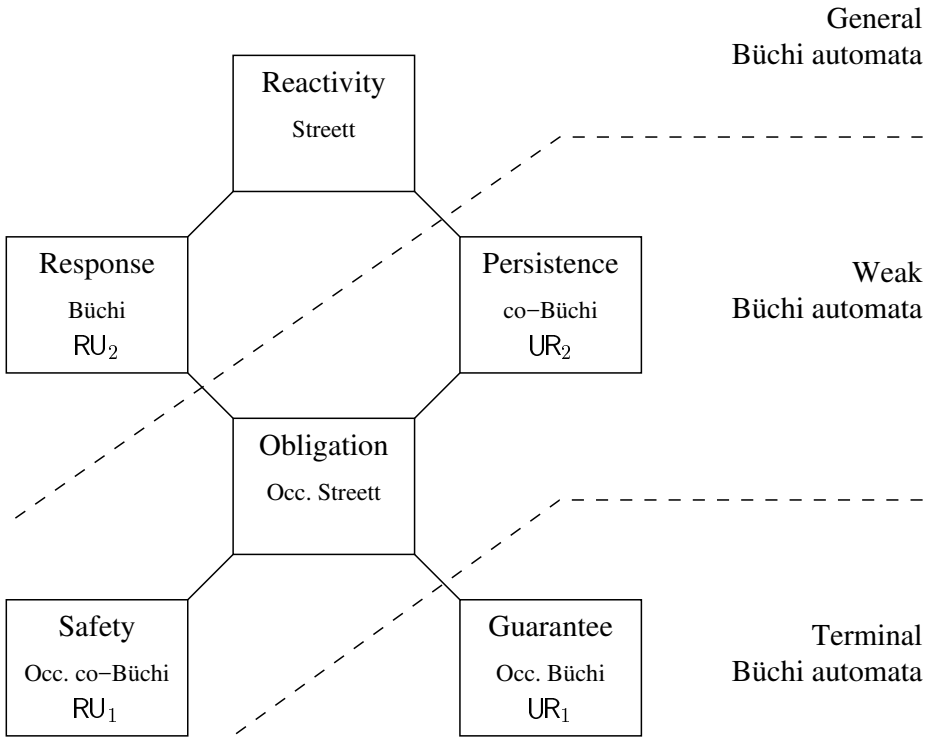

Figure 3.9: The hierarchy of temporal properties.

We now explain the relations between these classes and (canonical and future) formulae of corresponding types. All the following statements hold for every $\kappa \in \{$ safety, guarantee, obligation, response, persistence, reactivity $\}$.

Theorem 3.42 ([MP90b, CMP92]) *Let L be an LTL language of infinite words. The following statements are equivalent:*

1. *L is a κ property.*
2. *L is definable by a canonical κ formula.*
3. *L is definable by a future κ formula.*

Hence, every canonical κ fragment is expressively equivalent to the corresponding future κ fragment. Further, as every LTL language is known to be expressible in the future reactivity fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ (see Corollary 3.4), we get that every ω-language definable in LTL is a reactivity property. This can be also derived from the following theorem.

Theorem 3.43 ([MP90b]) *Every LTL formula is initially equivalent to a canonical reactivity formula.*

To sum up, every class of ω-languages definable by (canonical or future) κ fragment coincides with the corresponding class of κ properties restricted to LTL languages. The hierarchy in Figure 3.9 remains strict even under this restriction [MP90b].

The decidability of the considered LTL fragments is a consequence of the relations between the six property classes and six types of Streett automata.

Definition 3.44 *Let $A = (\Sigma, Q, q_0, \delta, \{(G_1, R_1), \ldots, (G_n, R_n)\})$ be a deterministic Street automaton such that $n = 1$. Let $G = (Q \setminus G_1) \cup R_1$ and $B = Q \setminus G$. For every $q, q' \in Q$ we write $(q, q') \in \delta$ instead of $\exists a \in \Sigma$ such that $q' \in \delta(q, a)$. The automaton A is*

a safety *automaton* *iff* $\forall q \in B, q' \in G : (q, q') \notin \delta$,

a guarantee *automaton* *iff* $\forall q \in G, q' \in B : (q, q') \notin \delta$,

an obligation *automaton* *iff* *there exists a function* $r : Q \to \{0, \ldots, k\}$ *such that:*
$(q, q') \in \delta \;\Rightarrow\; r(q) \leq r(q')$,
$(q \in B \wedge q' \in G \wedge (q, q') \in \delta) \;\Rightarrow\; r(q) < r(q')$,
$(q \in G \wedge q' \in B \wedge (q, q') \in \delta) \;\Rightarrow\; r(q) < k$,

a response *automaton* *iff* $G_1 = Q$,

a persistence *automaton* *iff* $R_1 = \emptyset$.

Further, every deterministic Street automaton[8] *is a* reactivity *automaton.*

Theorem 3.45 ([MP90b]) *An ω-regular language is a κ property if and only if it is recognized by a κ automaton.*

Theorem 3.46 ([MP90b]) *It is decidable whether an ω-regular language (given by a deterministic Streett automaton) is a κ property.*

The decidability of κ fragments then follows from the fact that every ω-language definable in LTL is ω-regular and from the decidability of LTL.

Corollary 3.47 *The canonical and future κ fragments are decidable over ω-words.*

The paper [MP90b] presents an outline of translations from canonical κ formulae to κ automata and from counter-free κ automata to κ formulae.

[8] Here the acceptance set is not restricted to the form $\{(G_1, R_1)\}$.

These translations are based on general constructions described in [LPZ85] and [Zuc86].

Another automata-based characterizations of κ properties have been suggested in [ČP03] and are indicated in Figure 3.9.

Theorem 3.48 ([ČP03]) *An ω-regular language is a safety, guarantee, obligation, response, persistence, or reactivity property if and only if it is recognized by a deterministic occurrence co-Büchi, deterministic occurrence Büchi, deterministic occurrence Streett, deterministic Büchi, deterministic co-Büchi, or deterministic Streett automaton respectively.*

3.4.2 Until-release hierarchy

The until-release hierarchy has been defined and studied in [ČP03]. The hierarchy has a strong connection to the hierarchy of temporal properties discussed above.

Theorem 3.49 ([ČP03]) *An ω-language definable by LTL is a safety, guarantee, response or persistence property if and only if it is definable by a formula of RU_1, UR_1, RU_2, or UR_2 respectively.*

The relations are depicted in Figure 3.9 too.

Further, the equality between LTL ω-languages and reactivity properties definable in LTL (see Theorem 3.43), the definition of reactivity property, and the previous theorem give us the following.

Theorem 3.50 ([ČP03]) *An ω-language is definable in LTL if and only if it is definable by a positive boolean combination of RU_2 and UR_2 formulae.*

This statement directly implies that the hierarchy of RU_i and UR_i fragments collapses.

Corollary 3.51 *An ω-language is definable in LTL if and only if it is definable by both RU_3 and UR_3.*

The decidability of RU_i and UR_i fragments over ω-words is then a direct consequence of the Corollary 3.47, Theorem 3.49, Corollary 3.51, and the decidability of LTL.

Corollary 3.52 *For every i, the fragments RU_i and UR_i are decidable over infinite words.*

Another interesting aspect of this hierarchy is its connection to terminal and weak Büchi automata. It is easy to prove that an LTL ω-language is a guarantee or persistence property if and only if it is recognized by a terminal or weak Büchi automaton respectively. Moreover, automata of these types can be constructed directly from formulae of the corresponding fragments UR_1 and UR_2.

Theorem 3.53 ([ČP03]) *For every ω-language given by a formula of UR_1 or UR_2 one can construct a terminal or weak Büchi automaton recognizing the language, respectively.*

The translation is a modification of the original construction of a Büchi automaton for a given $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y})$ formula introduced in [WVS83]. Similar modification has been presented in [Sch01] as well.

The theorem has some important consequences in the context of model checking. Recall that the property automaton (see Section 2.4) is a Büchi automaton corresponding to negation of a given specification formula. Thus, specification formulae of RU_1 (defining safety properties) can be translated to terminal property automata while specification formulae of RU_2 (defining response properties) can be translated to weak property automata. One can readily confirm that the type of a property automaton carries to the corresponding product automaton. Several specialized algorithms (explicit as well as symbolic) for non-emptiness check of weak and terminal Büchi automata have been proposed. The specialized symbolic algorithms have lower asymptotic complexity comparing to general symbolic non-emptiness algorithms. Asymptotic complexity of the specialized explicit algorithms is linear as in the case of general explicit algorithms. Nevertheless, the specialized algorithms have still several benefits including a possibility of employing some efficient heuristics, simpler implementations of partial order reductions, or possibility of better distribution. To stress the significance of the specialized algorithms, authors of [ČP03] have studied *specification patterns system* [DAC99] – a collection of the most often verified properties. They have calculated that safety properties (generating terminal property automata) comprise 41% and response properties (generating weak property automata) comprise 54%[9] of the properties in the collection. For more information about the specialized algorithms, their benefits, and discussion about the complexity of property type determination we refer to [ČP03].

Another characterization of the ω-languages definable in RU_i and UR_i fragments is given in Subsection 7.3.2. The characterization is formulated in terms of alternating 1-weak Büchi automata.

3.4.3 Deterministic fragment

The fragment detLTL have been introduced and studied in [Mai00] in the context of the properties (of Kripke structures) expressible in both LTL and CTL. As branching time logics are out of the scope of this thesis, we only give a brief and intuitive summary of the main results without providing any technicalities.

[9] The percentage refers to the response properties that are not safety properties.

In fact, the paper works just with ACTL, i.e. the fragment of CTL formulae which are in positive normal form and have no existential quantifiers. The fragment $\mathrm{detACTL}$ of ACTL is defined in a similar way as detLTL. The paper proves the following results.

- An ACTL formula is expressible[10] in LTL if and only if it is expressible in $\mathrm{detACTL}$.
- The problem whether a given ACTL formula is expressible in LTL or not is PSPACE-complete.
- An LTL formula is expressible in ACTL if and only if it is expressible in detLTL.

It is known that the logics CTL and LTL have incomparable expressive power. Hence, the results mentioned above imply that the expressive power of detLTL is strictly lesser than the expressive power of LTL. The same result can be achieved using the following characterization.

Theorem 3.54 ([Mai00]) *An LTL formula φ is expressible in detLTL if and only if there exists a 1-weak Büchi automaton recognizing the language $L(\neg\varphi)$.*

Moreover, for every detLTL formula φ there exists a 1-weak Büchi automaton of the size $\mathcal{O}(|\varphi|)$ recognizing the language $L(\neg\varphi)$.

The paper [Mai00] also provides several references to other papers dealing with relations between LTL and CTL.

3.4.4 Flat fragment

The fragment $\mathrm{flatLTL}(\mathsf{U})$ has been studied in [Dam99] (together with flat fragments of CTL and CTL*). It has been shown that $\mathrm{flatLTL}(\mathsf{U})$ is strictly less expressive than $\mathrm{LTL}(\mathsf{U})$.

Theorem 3.55 ([Dam99]) *There exists an $\mathrm{LTL}(\mathsf{U}^3)$ formula which is not equivalent to any formula of $\mathrm{flatLTL}(\mathsf{U})$. Moreover, every $\mathrm{LTL}(\mathsf{U}^2)$ formula is equivalent to a formula of $\mathrm{flatLTL}(\mathsf{U})$.*

Besides the expressive power, the paper studies also the *distinguishing* power of flat fragments.

Definition 3.56 *Let α, β be ω-words and $\mathcal{F}$ be an LTL fragment. The words are* distinguishable by fragment $\mathcal{F}$ *if there exists a formula φ of $\mathcal{F}$ such that*

$$\alpha \models \varphi \text{ if and only if } \beta \not\models \varphi.$$

[10] Here the expressibility is defined in a more general way; we say that a formula φ of CTL or LTL is expressible in a fragment/logic $\mathcal{F}$ if there exists a formula ψ of $\mathcal{F}$ such that for every Kripke structure K it holds that $K \models \varphi$ iff $K \models \psi$. The formal semantics of CTL can be found e.g. in [Eme90].

Theorem 3.57 ([Dam99]) *Let α, β be an ω-words. The words are distinguishable by* $\mathrm{LTL}(\mathsf{U})$ *if and only if they are distinguishable by* $\mathrm{flatLTL}(\mathsf{U})$.

Decidability of $\mathrm{flatLTL}(\mathsf{U})$ fragment is an open question.

3.5 Succinctness

The previous sections give an overview of expressiveness results related to LTL and its fragments. In the following we present several results comparing formalisms with respect to their succinctness instead of the raw expressive power.

In Section 3.1 we argue that there is no reason to distinguish between LTL with modalities U and X and its syntactic supersets as they have the same expressive power. However, their expressive power is different when the succinctness is taken into account.

In the light of the fact that $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ and $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ are expressively equivalent and with respect to the syntactic algorithm [Gab89] translating a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ into an initially equivalent formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$, the past modalities are often dropped from the logic. However, there are good reasons not to do so. First, the satisfiability and model-checking are PSPACE-complete for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ as well as for $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ [SC85] (for more information about complexity issues see Chapter 4). Further, some specification formulae are easier to write using the past modalities as well [LPZ85]. This statement is formally justified by the following theorem. The undefined term 'exponentially more succinct' will be explained by the comment below the theorem.

Theorem 3.58 ([LMS02, Mar03b]) $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ *can be exponentially more succinct than* $\mathrm{LTL}(\mathsf{U},\mathsf{X})$.

The proof introduces a parametrized $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formula ψ_n of the size $\mathcal{O}(n)$ such that the size of an initially equivalent formula φ_n of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ is in $\Omega(2^n)$. Moreover, the formula φ_n has $\Omega(2^n)$ distinct subformulae, so it cannot be succinctly represented as a logical circuit (DAG). Further, ψ_n is in fact a formula of $\mathrm{LTL}(\mathsf{F},\mathsf{P},\mathsf{Y})$ and authors of the statement say that the proof could be adapted to use a formula of $\mathrm{LTL}(\mathsf{F},\mathsf{P})$.

Theorem 3.58 gives a lower bound of the succinctness gap. The upper bounds are given by translations of $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formulae into $\mathrm{LTL}(\mathsf{U},\mathsf{X})$. The size of the formulae produced by the Gabbay's translation [Gab89] is assumed to be nonelementary (it has not been characterized precisely so far). Better upper bound has been presented in [Mar03b]; using the results of [LPZ85, MP90a, MP94] a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ can be translated into an initially equivalent $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula of (at most) triply exponential size.

To sum up, the succinctness gap between LTL with future modalities and LTL with both future and past modalities is at least single exponential and at most triply exponential. The precise characterization of this succinctness gap is an open question.

The modality N can further improve the succinctness of $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$.

Theorem 3.59 ([LMS02]) $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y},\mathsf{N})$ *can be exponentially more succinct than* $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$.

Let us note that this is only a lower bound.

The following results compare the succinctness of LTL and other formalism, namely first-order logic and automata over finite or infinite words.

Every LTL formula can be translated into a globally equivalent FOMLO formula of a linear size. However, FOMLO can be much more succinct than LTL.

Theorem 3.60 ([Mey75, Sto74]) *FOMLO can be nonelementarily more succinct than* $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$.

To be more specific, Stockmeyer [Sto74] defines a sequence (φ_n) of FO^3 formulae such that the size of φ_n is linear with respect to n and every sequence (ψ_n) of initially equivalent $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formulae satisfies $|\psi_n| \in \mathrm{tower}(\Omega(n/\log n))$, where $\mathrm{tower}(k)$ denotes the "tower" $2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ of the height k. The theorem can be also proved using the fact that the satisfiability problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ is PSPACE-complete while the satisfiability for FOMLO is nonelementary [SM73].

In contrast to this result, formulae of FO^2 can be translated into exponentially larger formulae of $\mathrm{LTL}(\mathsf{X},\mathsf{Y},\mathsf{F},\mathsf{P})$. Moreover, this translation is essentially optimal. In the following theorems $\mathrm{qdp}(\varphi)$ denotes the quantifier depth of a FOMLO formula φ and $\mathrm{odp}(\psi)$ denotes the nesting depth of all temporal operators in an LTL formula ψ.

Theorem 3.61 ([EVW02]) *Every* FO^2 *formula* $\varphi(x)$ *can be translated into a globally equivalent* $\mathrm{LTL}(\mathsf{F},\mathsf{X},\mathsf{P},\mathsf{Y})$ *formula* ψ *of the size* $2^{\mathcal{O}(|\varphi|(\mathrm{qdp}(\varphi)+1))}$ *such that* $\mathrm{odp}(\psi) \leq 2\mathrm{qdp}(\varphi)$.

Theorem 3.62 ([EVW02]) *There is a sequence* (φ_n) *of* FO^2 *formulae with one propositional variable such that the size of* φ_n *is* $\mathcal{O}(n^2)$ *and every sequence* (ψ_n) *of initially equivalent* $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ *formulae satisfies* $|\psi_n| \in 2^{\Omega(n)}$.

Similar results hold for $\mathsf{FO}^2[<]$ and $\mathrm{LTL}(\mathsf{F}_\mathsf{s},\mathsf{P}_\mathsf{s})$.

Theorem 3.63 ([EVW02]) *Every* $\mathsf{FO}^2[<]$ *formula* $\varphi(x)$ *can be translated into a globally equivalent* $\mathrm{LTL}(\mathsf{F}_\mathsf{s},\mathsf{P}_\mathsf{s})$ *formula* ψ *of the size* $2^{\mathcal{O}(|\varphi|(\mathrm{qdp}(\varphi)+1))}$ *such that* $\mathrm{odp}(\psi) \leq \mathrm{qdp}(\varphi)$.

Theorem 3.64 ([EVW02]) *There is a sequence (φ_n) of $\mathsf{FO}^2[<]$ formulae such that the size of φ_n is $\mathcal{O}(n)$ and every sequence (ψ_n) of initially equivalent $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formulae satisfies $|\psi_n| \in 2^{\Omega(n)}$.*

Finally, we present some succinctness results regarding finite automata.

Theorem 3.65 ([VW94]) *Every ω-language defined by an $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ formula φ is recognized by a (nondeterministic) Büchi automaton with $2^{\mathcal{O}(|\varphi|)}$ states.*

This upper bound on the size of the Büchi automaton is known to be tight.

The translation of deterministic counter-free automata over finite words into LTL given in [Wil99] yields the following upper bound.

Theorem 3.66 ([Wil99]) *For every counter-free deterministic finite automaton over finite words there exists an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula φ such that φ defines the language recognized by the automaton and the size of φ is $|\Sigma| \cdot 2^{2^{\mathcal{O}(n \log n)}}$, where n is the number of states of the automaton and Σ is its input alphabet.*

3.6 Additional notes

The overview of expressiveness, decidability, and succinctness results presented in this Chapter is far from being complete. For example, we do not mention connections between some of the presented results and the *dot-depth hierarchy* (discussed for example in [TW98, Sch00]). We also omit the definition and characterizations of *liveness* properties (see [CMP92] for some references).

Many natural questions concerning the expressiveness of LTL fragments are still open. Some of them were formulated several years ago, for example:

- Can the decidability results based on forbidden patterns be adapted for ω-languages?
- Is the fragment $\mathrm{flatLTL}(\mathsf{U})$ decidable?
- What is the complexity of the problem whether a language given by an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula is expressible in $\mathrm{LTL}(\mathsf{F}_\mathsf{s},\mathsf{P}_\mathsf{s})$? And what is the complexity of the analogous problem for the fragment $\mathrm{LTL}(\mathsf{F},\mathsf{X},\mathsf{P},\mathsf{Y})$? See Theorem 3.29.

These questions have been formulated in [EW00], [Dam99], and [TW98] respectively. Some questions have not been considered before and they may be easy to solve, for example:

- Are the fragments $\mathrm{LTL}(\overset{\infty}{\mathsf{F}})$ and $\mathrm{LTL}(\overset{\infty}{\mathsf{F}},\mathsf{X})$ decidable?

- What is the complexity of the problem whether a language given by a nondeterministic Büchi automaton is expressible in LTL?

Further, numerous topics for future research can be found in the area of nesting fragments as only several types of these fragments have been studied.

Finally, there are many open questions about succinctness of various LTL fragments. The most prominent one is the precise succinctness of $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y})$ in a comparison with $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ (see Section 3.5).

Chapter 4

Complexity issues

In this chapter we deal with satisfiability, model checking, and model checking a path problems for various fragments of LTL. As all these problems are decidable[1], we focus on complexity[2] of these problems. The chapter summarizes the results established or cited in [SC85, LP85, LPZ85, DS02, LMS02, EVW02, Mar04, MS03, Sch03]. The problems are considered in context of infinite words or paths.

First of all we define these three problems more precisely. Let $\mathcal{F}$ be a fragment of LTL.

Satisfiability. Satisfiability problem for $\mathcal{F}$ is the problem to decide whether for a given formula $\varphi \in \mathcal{F}$ there exists an ω-word α such that $\alpha \models \varphi$.

Model checking. In fact, two versions of this problem are considered in literature. Up to now we have been discussing the problem whether a given formula $\varphi \in \mathcal{F}$ and a given Kripke structure K satisfy $K \models \varphi$, i.e. whether φ is valid for all paths in the Kripke structure. This problem is called the *universal model checking* for $\mathcal{F}$.

Let $K = (S, T, s_I, L)$ be a Kripke structure and φ a formula. We write $K \models^{\exists} \varphi$ if there exists a path $s_0, s_1, \ldots$ in K such that $s_0 = s_I$ and $L(s_0)L(s_1)\ldots \models \varphi$. The *existential model checking* for $\mathcal{F}$ is the problem to decide whether for a given Kripke structure K and a formula $\varphi \in \mathcal{F}$ it holds that $K \models^{\exists} \varphi$, i.e. whether φ is valid at least for one path in the Kripke structure.

Model checking a path. Model checking a path for $\mathcal{F}$ is the problem to decide whether a given formula $\varphi \in \mathcal{F}$ and given finite words u, v (where $v \neq \varepsilon$) satisfy $uv^\omega \models \varphi$. In [MS03], the complexity of analogous problem for finite words is studied as well.

[1] We consider model checking of finite-state systems only.

[2] We assume that the reader is familiar with basic terms of computational complexity theory. We refer to books [Sip97, Pap94] for an introduction to this theory.

Research focused on computational complexity of LTL model checking is usually working with existential version of the problem as it is closer to the satisfiability problem. However, the complexity of universal model checking can be derived from the complexity results on existential model checking as these two problems are dual in the following sense:

$$K \models^{\exists} \varphi \text{ if and only if } K \not\models \neg\varphi$$

Let $\mathcal{F}$ be a fragment closed under negation and existential model checking for $\mathcal{F}$ be in a complexity class C. Universal model checking for the fragment is then in complexity class $\mathsf{co}-C$. If $\mathcal{F}$ is not closed under negation, the complexity of universal model checking for $\mathcal{F}$ can be derived from the complexity of existential model checking for the fragment of negated $\mathcal{F}$ formulae. For example, universal model checking for $\mathrm{LTL}^+(\mathsf{G},\mathsf{X})$ is coNP-complete as existential model checking for $\mathrm{LTL}^+(\mathsf{F},\mathsf{X})$ is NP-complete. In the following we concentrate on existential version of the model checking problem.

First, we sum up the complexity results for satisfiability and existential model checking. Then we mention the complexity results of model checking a path. Together with various LTL fragments, the FOMLO fragments FO^2 and $\mathsf{FO}^2[<]$ are also studied in this chapter.

4.1 Satisfiability and model checking

Complexity of satisfiability and model checking problems for LTL fragments was systematically studied for the first time by Sistla and Clark in a conference version of [SC85] published in 1982. The paper shows that while these problems are NP-complete for $\mathrm{LTL}(\mathsf{F})$ and $\mathrm{LTL}^+(\mathsf{F},\mathsf{X})$, they are PSPACE-complete for $\mathrm{LTL}(\mathsf{F},\mathsf{X})$, $\mathrm{LTL}(\mathsf{U})$, $\mathrm{LTL}(\mathsf{U},\mathsf{X})$, and $\mathrm{LTL}(\mathsf{U},\mathsf{S},\mathsf{X})$ (and also for the extended temporal logic (ETL) [Wol83]). The PSPACE upper bounds for existential model checking are achieved due to the following theorem.

Theorem 4.1 ([SC85]) *Let $\mathcal{F}$ be a fragment subsuming $\mathrm{LTL}(\mathsf{F},\mathsf{X})$ or $\mathrm{LTL}(\mathsf{U})$. Existential model checking for $\mathcal{F}$ is polynomial-time reducible to the satisfiability problem for $\mathcal{F}$.*

The proof of PSPACE-completeness of the two problems for $\mathrm{LTL}(\mathsf{U},\mathsf{S},\mathsf{X})$ can be adapted for the fragment $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$. A different algorithm deciding the satisfiability problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ with a better time complexity has been presented in [LPZ85].

In spite of PSPACE-hardness, model checking of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ properties has proved to be feasible in practice. Several explanations of this fact has been provided. A crucial observation formulated in [LP85] says that model checking is only linear in the size of a Kripke structure.

Theorem 4.2 ([LP85, VW86]) *Given a Kripke structure $K = (S, T, s_I, L)$ and a formula $\varphi \in \mathrm{LTL}(\mathsf{U}, \mathsf{X})$, the model checking problem can be solved in time $\mathcal{O}(|K|) \cdot 2^{\mathcal{O}(|\varphi|)}$, where $|K| = |S| + \Sigma_{t \in T}|t|$ is the number of states and edges in the structure.*

The exponential time complexity in the size of a specification formula is often annotated with arguments that specification formulae are usually short and have very low nesting depths of modalities. Inspired by this argumentation, Demri and Schnoebelen [DS02] have studied the theoretical complexity of satisfiability and model checking for fragments with low nesting depth of all temporal modalities and/or bounds on the number of atomic propositions occurring in a formula. Their results are rather negative in the sense that the complexity of satisfiability and model checking for fragments $\mathrm{LTL}(\mathsf{F})$, $\mathrm{LTL}(\mathsf{U})$, and $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ remains unchanged when we restrict the number of atomic propositions to two or bind the nesting depth of all temporal operators by two. On the other hand, the paper shows that model checking for $\mathrm{LTL}(\{\mathsf{F}, \mathsf{X}\}^k)$ is NP-complete while the same problem is PSPACE-complete for $\mathrm{LTL}(\mathsf{F}, \mathsf{X})$. For more results see Table 4.1. The paper develops several logspace reductions of satisfiability, 3SAT, and QBF problems to the model checking problem for the fragments studied there. The flat versions of the considered fragments are studied as well. It is proven that the flattening has no influence on the complexity of satisfiability and model checking.

We have already mentioned that the addition of past modalities S, Y to the fragment $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ does not change complexities of the considered problems. This observation brings a question whether allowing past counterparts of included future modalities has no influence on complexities of the considered problems in general. The results presented in [Mar04] support an assumed positive answer. The paper studies the complexity of satisfiability and model checking for $\mathrm{LTL}(\mathsf{F}, \mathsf{P})$, $\mathrm{LTL}(\mathsf{X}, \mathsf{S}, \mathsf{Y})$, and positive (and stratified) fragments built with future and past modalities. For more results see Tables 4.1 and 4.2. The paper [Mar04] provides also complexities of universal model checking for the considered fragments.

Introducing the modality N the paper [LMS02] also demonstrates that satisfiability and model checking for $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{S}, \mathsf{Y}, \mathsf{N})$ are EXPSPACE-complete. The influence of the temporal operator C on the complexity of the considered problems has been studied in [RP86]. Both satisfiability and model checking for $\mathrm{LTL}(\mathsf{U}, \mathsf{X}, \mathsf{C})$ are nonelementary.

The complexity of satisfiability problem for fragments FO^2, $\mathsf{FO}^2[<]$, and $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$ has been studied in [EVW02]. Surprisingly, the problem is NEXP-complete for FO^2 and $\mathsf{FO}^2[<]$ (while it is nonelementary for FO^3), and NP-complete for $\mathrm{LTL}(\mathsf{F_s}, \mathsf{P_s})$. We now analyse the complexity of existential model checking for these fragments.

Theorem 4.3 *Existential model checking for FO^2 is NEXP-complete.*

Proof: The existential model checking problem for FO^2 is polynomial-time reducible to the satisfiability problem for FO^2. The reduction is a straightforward analogy of the reduction for $\mathrm{LTL}(\mathsf{F},\mathsf{X})$ described in [SC85] (see Theorem 4.1). This gives us the upper bound.

We show that the existential model checking problem is NEXP-hard even for the fragment $\mathsf{FO}^2_1[suc]$ which is smaller than FO^2. The fragment $\mathsf{FO}^2_1[suc]$ consists of all FOMLO formulae with at most two variables, one unary predicate, and without the predicate $<$. The NEXP-hardness of existential model checking for $\mathsf{FO}^2_1[suc]$ is a direct consequence of the following two facts.

1. The satisfiability problem for $\mathsf{FO}^2_1[suc]$ is NEXP-hard [EVW02].

2. The satisfiability problem for a FOMLO fragment with the bounded number of unary predicates is polynomial-time reducible to the existential model checking problem for the fragment.[3] Intuitively, given a FOMLO formula with n unary predicates $P_0, P_1, \ldots, P_{n-1}$ the reduction creates a Kripke structure with paths corresponding to all ω-words over alphabet $2^{\{p_0,p_1,\ldots,p_{n-1}\}}$. Hence, the formula is satisfiable if and only if it is valid for at least one path in the structure. Moreover, the Kripke structure has 2^n states and thus its size depends only on n. ■

Corollary 4.4 *Existential model checking for* $\mathsf{FO}^2[<]$ *is in* NEXP.

Theorem 4.5 *Existential model checking for* $\mathrm{LTL}(\mathsf{F}_s,\mathsf{P}_s)$ *is* NP*-complete.*

Proof: The upper bound can be obtained by a slight modification of analogous result for $\mathrm{LTL}(\mathsf{F},\mathsf{P})$ given in [Mar04]. The lower bound is a consequence of NP-hardness of existential model checking for $\mathrm{LTL}(\mathsf{F})$ established in [SC85]. ■

Tables 4.1 and 4.2 give an overview of the complexity results obtained or cited in the papers described above. The prefix '(flat)' before a fragment means that the complexity of satisfiability and model checking is the same for the flat version of the fragment too. The tables are arranged according to the modalities used in the fragments; Table 4.1 covers the fragments with future modalities only while the other fragments (together with ETL and some FOMLO fragments) are contained in Table 4.2.

Let us note that complexities of satisfiability and existential model checking for many fragments not covered by the tables can be easily derived from the results summarized therein. For example, the two problems

[3] An analogous reduction for LTL fragments with the bounded number of atomic propositions is given in [DS02].

Fragment ($1 \leq n, k < \omega$)	**Satisfiability**	**Existential MC**
$\mathrm{LTL}()$	NP-complete	L
$\mathrm{LTL}_n()$	L	L
$\mathrm{LTL}(\overset{\infty}{\mathsf{F}})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{F})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{F}^1)$	NP-complete	NP-complete
$\mathrm{LTL}_2(\mathsf{F})$	NP-complete	NP-complete
$\mathrm{LTL}_1(\mathsf{F})$	P	in P, NL-hard
$\mathrm{LTL}_n(\mathsf{F}^k)$	L	NL-complete
$\mathrm{LTL}^+(\mathsf{F})$	NP-complete	NP-complete
$\mathrm{LTL}^+(\mathsf{G})$	NP-complete	NP-complete
$(\mathrm{flat})\mathrm{LTL}(\mathsf{U})$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}(\mathsf{U}^2)$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}(\mathsf{U}^1)$	NP-complete	NP-complete
$(\mathrm{flat})\mathrm{LTL}_2(\mathsf{U})$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}_1(\mathsf{U})$	P	in P, NL-hard
$(\mathrm{flat})\mathrm{LTL}_n(\mathsf{U}^k)$	L	NL-complete
$\mathrm{LTL}^+(\mathsf{U})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}(\mathsf{X})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{X}^k)$	NP-complete	L
$\mathrm{LTL}_1(\mathsf{X})$	NP-complete	NP-complete
$\mathrm{LTL}_n(\mathsf{X}^k)$	L	L
$\mathrm{LTL}^+(\mathsf{X})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{F}, \mathsf{X})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}(\{\mathsf{F}, \mathsf{X}\}^{1+k})$	PSPACE-complete	NP-complete
$\mathrm{LTL}(\{\mathsf{F}, \mathsf{X}\}^1)$	NP-complete	NP-complete
$\mathrm{LTL}_1(\mathsf{F}, \mathsf{X})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}_n(\{\mathsf{F}, \mathsf{X}\}^k)$	L	NL-complete
$\mathrm{LTL}^+(\mathsf{F}, \mathsf{X})$	NP-complete	NP-complete
$\mathrm{LTL}^+(\mathsf{G}, \mathsf{X})$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}(\mathsf{U}, \mathsf{X})$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}(\{\mathsf{U}, \mathsf{X}\}^2)$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}(\{\mathsf{U}, \mathsf{X}\}^1)$	NP-complete	NP-complete
$(\mathrm{flat})\mathrm{LTL}_1(\mathsf{U}, \mathsf{X})$	PSPACE-complete	PSPACE-complete
$(\mathrm{flat})\mathrm{LTL}_n(\{\mathsf{U}, \mathsf{X}\}^k)$	L	NL-complete
$\mathrm{LTL}(\{\mathsf{U}, \overset{\infty}{\mathsf{F}}, \mathsf{X}\}^1)$	NP-complete	NP-complete

Table 4.1: Complexity of satisfiability and existential model checking.

Fragment/Logic	Satisfiability	Existential MC
$\mathrm{LTL}(\mathsf{F},\mathsf{P})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{F}_s,\mathsf{P}_s)$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{X},\mathsf{S},\mathsf{Y})$	NP-complete	NP-complete
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}^{+s}(\mathsf{F},\mathsf{Y})$	NP-complete	NP-complete
$\mathrm{LTL}^{+s}(\mathsf{G},\mathsf{Y})$	NP-complete	PSPACE-complete
$\mathrm{LTL}^{+}(\mathsf{F},\mathsf{X},\mathsf{P},\mathsf{Y})$	NP-complete	NP-complete
$\mathrm{LTL}^{+}(\mathsf{G},\mathsf{X},\mathsf{H},\mathsf{Y})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}^{+s}(\mathsf{G},\mathsf{S},\mathsf{Y})$	NP-complete	PSPACE-complete
$\mathrm{LTL}^{+}(\mathsf{G},\mathsf{S},\mathsf{Y})$	NP-complete	PSPACE-complete
$\mathrm{LTL}^{+s}(\mathsf{F},\mathsf{S})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}^{+s}(\mathsf{G},\mathsf{S})$	NP-complete	PSPACE-complete
$\mathrm{LTL}^{+}(\mathsf{G},\mathsf{S})$	NP-complete	PSPACE-complete
$\mathrm{LTL}^{+}(\mathsf{U},\mathsf{S})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}^{+}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$	PSPACE-complete	PSPACE-complete
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y},\mathsf{N})$	EXPSPACE-complete	EXPSPACE-complete
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{C})$	nonelementary	nonelementary
ETL	PSPACE-complete	PSPACE-complete
$\mathsf{FO}^2[<]$	NEXP-complete	NEXP
FO^2	NEXP-complete	NEXP-complete
FO^3	nonelementary	nonelementary
FOMLO	nonelementary	nonelementary

Table 4.2: Complexity of satisfiability and existential model checking.

are NP-complete for $\mathrm{LTL}(\mathsf{F}_s)$ as the fragment subsumes $\mathrm{LTL}(\mathsf{F})$, it is subsumed in $\mathrm{LTL}(\mathsf{F}_s,\mathsf{P}_s)$, and the problems are NP-complete for both $\mathrm{LTL}(\mathsf{F})$ and $\mathrm{LTL}(\mathsf{F}_s,\mathsf{P}_s)$.

For more information on the complexity results of model checking for LTL, CTL, and CTL* (and their fragments) we refer to a nicely written summary [Sch03]. This paper contains all the basic definitions, theorems, and proof techniques used in the area together with references to many related papers. Besides the results discussed in the paragraphs above, the paper also deals with complexities of the model checking problem for $\mathrm{LTL}(\overset{\infty}{\mathsf{F}})$ and $\mathrm{LTL}(\{\mathsf{U},\overset{\infty}{\mathsf{F}},\mathsf{X}\}^1)$, and the complexity of symbolic model checking (where the size of Kripke structure is defined as a sum of sizes of its parallel components).

The paper [Sch03] also recapitulates some basic results on program-complexity and formula-complexity of model checking. The last two terms have been defined in [VW86]. The *program-complexity* of model checking is

its complexity measured as a function of the size of a Kripke structure only. By analogy, the *formula-complexity* is measured in the size of a specification formula only.

The program-complexity is connected with NL-complete problem whether a given state is reachable in a given structure. The program-complexity of existential model checking for LTL with all the modalities defined so far is in NL as the problem reduces to reachability in the corresponding product automaton. The reachability problem reduces to the existential model checking problem for all the fragments subsuming $\mathrm{LTL}^{+}(\mathsf{F}^{1})$. Hence, the program-complexity for these fragments is NL-hard.

The formula-complexity is not higher than the (standard) complexity. Further, for every fragment defined over a finite set of atomic propositions $P \subseteq At$, the satisfiability problem reduces to existential model checking where the Kripke structure is fixed and generates all infinite words over 2^{P}. As satisfiability of $\mathrm{LTL}_1(\mathsf{F},\mathsf{X})$ or $\mathrm{LTL}_2(\mathsf{U})$ is PSPACE-complete, the formula-complexity for all the fragments subsuming $\mathrm{LTL}_1(\mathsf{F},\mathsf{X})$ or $\mathrm{LTL}_2(\mathsf{U})$ is PSPACE-hard. To sum up, the formula-complexity for all the fragments subsuming $\mathrm{LTL}_1(\mathsf{F},\mathsf{X})$ or $\mathrm{LTL}_2(\mathsf{U})$ and subsumed in $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ is PSPACE-complete. The formula-complexity of existential model checking for $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y},\mathsf{N})$ is EXPSPACE-complete [LMS02].

4.2 Model checking a path

Roughly speaking, the previous section shows that the model checking problem for all LTL fragments with "reasonable" expressive power is PSPACE-hard. In contrast, *model checking a path* represents an interesting subproblem with significantly lower asymptotic complexity. The problem has been identified only recently by Markey and Schnoebelen in [MS03].

We recall that model checking a path is model checking of finite Kripke structures with a single path. For brevity of notation, instead of Kripke structures defining one (infinite) path we work directly with words of the form uv^{ω}, where $u, v \in \Sigma^{*}$ and $v \neq \varepsilon$. As CTL interpreted over linear structures coincides with LTL, the problem whether a given formula $\varphi \in \mathrm{LTL}(\mathsf{U},\mathsf{X})$ and a given words u, v satisfy $uv^{\omega} \models \varphi$ can be solved in time $\mathcal{O}(|uv| \cdot |\varphi|)$ using the standard algorithm for CTL model checking of finite-state systems [CES86].

The paper [MS03] studies the same problem for other three LTL fragments and FOMLO as well. Table 4.3 provides an overview of the complexity results presented in the paper. For all the fragments occurring in the table, the complexities in the middle column correspond to both existential and universal model checking problems.

For some specialized algorithms solving the problem, more accurate evaluations of their complexity, and analogous results for model check-

Fragment/Logic	Model checking	Model checking a path
$\mathrm{LTL}(\mathsf{U},\mathsf{X})$	PSPACE-complete	in P
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$	PSPACE-complete	in P
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y},\mathsf{N})$	EXPSPACE-complete	P-complete
$\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{C})$	nonelementary	P-complete
FOMLO	nonelementary	PSPACE-complete

Table 4.3: Complexity of model checking and model checking a path.

ing of compressed paths (defined by grammars) and finite paths, we refer to [MS03].

Another algorithm solving the problem of model checking a path for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae is presented in Subsection 6.2.2.

4.3 Additional notes

Some of the papers cited in this chapter also deal with the complexity of validity and universal model checking, or provide more detailed complexity analyses.

The subject area of this chapter still suggests several open questions. The most prominent ones correspond to the lines in Tables 4.1, 4.2, and 4.3 where the lower bounds do not match the upper bounds. More precisely, open questions are whether the existential model checking problems for $\mathrm{LTL}_1(\mathsf{F})$, $\mathrm{LTL}_1(\mathsf{U})$, and $\mathrm{flatLTL}_1(\mathsf{U})$ are P-complete or they are in a lower complexity class. Further, the paper [EVW02] contains a question whether satisfiability for $\mathsf{FO}^2_n[<]$ (fragment of $\mathsf{FO}^2[<]$ formulae with at most n unary predicates) remains NEXP-hard. A positive answer would imply NEXP-hardness of existential model checking for $\mathsf{FO}^2[<]$. The most important open questions regarding model checking a path are connected with the complexity of this problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ and $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$. The best known algorithms work in polynomial time, but there are no reasonable lower bounds. The authors of [MS03] mention that they have been unable to prove even L-hardness for $\mathrm{LTL}(\mathsf{U},\mathsf{X},\mathsf{S},\mathsf{Y})$ or to find a better (memory-efficient or parallel) algorithm even for model checking of finite paths for $\mathrm{LTL}(\mathsf{F})$. However, they conjecture that the problems are not P-hard.

Finally, we note that there are still some interesting LTL fragments which have not been studied in context of the satisfiability and model checking problems at all. For example, the complexity of these problems for $\mathrm{LTL}(\overset{\infty}{\mathsf{F}},\mathsf{X})$ can be of a particular interest as this fragment subsumes $\mathrm{LTL}(\mathsf{X})$ and $\mathrm{LTL}(\overset{\infty}{\mathsf{F}})$ (where these problems are NP-complete) and is subsumed in $\mathrm{LTL}(\mathsf{F},\mathsf{X})$ (where these problems are PSPACE-complete).

Chapter 5

Stuttering principles

This chapter presents the results originally introduced in papers [KS02, KS04, KS05b].

More than twenty years ago Lamport [Lam83b] argued that the validity of $\mathrm{LTL}(\mathsf{U})$ formulae does not depend on numbers of adjacent copies of letters in a word. This observation is called *stuttering principle*. Later this principle became one of the cornerstones of partial order reduction methods. In the following section we recall the stuttering principle and extend it to general $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae. Roughly speaking, the *general stuttering theorem* says that under certain 'local-periodicity' conditions (which depend on m and n) one can remove a given subword u from a given word α without influencing the (in)validity of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae (we say that u is (m, n)*-redundant* in α). In order to give some intuition for general stuttering, we define other two stuttering principles: *letter stuttering* (also known as n*-stuttering*) connected with nesting depth of X operators, and *subword stuttering* associated with formulae without X operator and with given bound on nesting depth of U operator.

Peled and Wilke [PW97a] have proved that if a language definable by LTL is *closed under stuttering* (i.e. it does not distinguish between words that are the same if we remove all adjacent copies of letters in the words) then the language can be defined even by an LTL formula without any X operator. In Section 5.2 we prove that every LTL language closed under n-stuttering can be defined in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$. Further, we show that general stuttering can be used for characterization of ω-regular languages that are definable in LTL.

Section 5.3 provides several theoretical applications of general stuttering theorem. The section addresses some basic problems regarding three hierarchies of LTL fragments, namely $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. In particular, the following problems seem to be among the most natural ones:

Question 1 Are those hierarchies semantically strict? That is, if we increase

m or n just by one, do we always obtain a strictly more expressive fragment of LTL?

Question 2 If we take two fragments $\mathcal{F}, \mathcal{F}'$ in the above hierarchies which are syntactically incomparable (for example, we can consider $\mathrm{LTL}(\mathsf{U}^4, \mathsf{X}^3)$ and $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^5)$, or $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X}^0)$ and $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X})$), are they also semantically incomparable? That is, are there formulae $\varphi_{\mathcal{F}} \in \mathcal{F}$ and $\varphi'_{\mathcal{F}} \in \mathcal{F}'$ such that $L(\varphi_{\mathcal{F}})$ is not expressible in $\mathcal{F}'$ and $L(\varphi'_{\mathcal{F}})$ is not expressible in $\mathcal{F}$?

Question 3 In the case of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchy, what is the semantic intersection of $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$? That is, what languages are expressible in both fragments?

We provide (positive) answers to Question 1 and Question 2. Here the results on the $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchy seem to be particularly interesting. As for Question 3, one is tempted to expect the following answer: The semantic intersection of $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ is exactly the set of languages expressible in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, where $m = \min\{m_1, m_2\}$ and $n = \min\{n_1, n_2\}$. Surprisingly, this answer turns out to be *incorrect*. For all $m \geq 1$, $n \geq 0$ we give an example of a language L which is definable both in $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n+1})$, but not in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. This shows that the answer to Question 3 is not as simple as one might expect. It is worth mentioning here that the class of languages expressible in both fragments $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ is decidable as these fragments are decidable (see Subsection 3.3.4). In fact, we are looking for an answer providing us better insight into expressiveness of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragments.

The results concerning Question 1 are closely related to the work of Etessami and Wilke [EW00]. They prove the strictness of *until hierarchy* (see Subsection 3.3.1) in the following way: First, they design an appropriate Ehrenfeucht-Fraïssé game for LTL (the game is played on a pair of words) which in a sense characterizes those pairs of words that can be distinguished by LTL formulae where the temporal operators are nested only to a certain depth. Then, for every $k \geq 1$ they construct a language FAIR_{k+1} definable by a $\mathrm{LTL}(\mathsf{U}^k, \mathsf{F}, \mathsf{X})$ formula and prove that this particular language cannot be expressed by any formula from $\mathrm{LTL}(\mathsf{U}^{k-1}, \mathsf{F}, \mathsf{X})$. Here the previous results about the designed EF game are used. Since the formula defining FAIR_{k+1} contains just one F operator and many nested X and U operators, this proof carries over to our $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$ hierarchy. In fact, [EW00] is a 'stronger' result in the sense that one additional nesting level of U cannot be 'compensated' by arbitrarily-deep nesting of X and F. On the other hand, the proof does not allow to conclude that, e.g. $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X}^0)$ contains a formula which is not expressible in $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X})$ (because the formula defining FAIR_{k+1} contains the nested X modalities).

Our method for solving Questions 1 and 2 is different. The general stuttering theorem gives us a simple (but surprisingly powerful) tool allowing to prove that a certain formula φ is *not* definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. The principle is applied as follows: we choose a suitable alphabet Σ, consider the language $L^\Sigma(\varphi)$, and find an appropriate $\alpha \in L^\Sigma(\varphi)$ and its subword u such that

- u is (m, n)-redundant in α;
- $\alpha' \not\models \varphi$ where α' is obtained from α by deletion of the subword u.

If we manage to do that, we can conclude that φ is not expressible in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.

Using this tool, the proofs of answers to Questions 1 and 2 are remarkably short though it took us some time to find appropriate formulae which witness the presented claims. It is worth noting that some of the known results about LTL (e.g. the formula '$\mathsf{G}_2 a$' is not definable in LTL) admit a one-line proof if our general stuttering principle is applied.

In section 5.4 we indicate that the general stuttering principle has the potential to improve the partial order reduction methods.

The last section of this chapter contains additional notes about stuttering including the situation on finite words.

5.1 A general stuttering theorem

In this section we formulate and prove the promised general stuttering theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages. General stuttering combines and extends two independent principles of *letter stuttering* (n-stuttering) and *subword stuttering*, which are applicable to the $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ fragments of LTL, respectively. We start by explaining these two principles in Section 5.1.1 and Section 5.1.2. This material has been included for two reasons. First, the two simplified principles are interesting on their own. In Section 5.2.1 we present special results about letter stuttering which do not hold for general stuttering. Secondly, the remarks and proof sketches given in Section 5.1.1 and Section 5.1.2 should help the reader in gaining some intuition about the functionality and underlying principles of general stuttering.

5.1.1 Letter stuttering (n-stuttering)

Letter stuttering is a simple generalization of the well-known principle of *stutter invariance* of $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^0)$ formulae [Lam83b] saying that $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^0)$ formulae cannot distinguish between one and more adjacent occurrences of the same letter in a given word. Formally, a letter $\alpha(i)$ of an ω-word α is called *redundant* if $\alpha(i) = \alpha(i+1)$ and there is $j > i$ such that $\alpha(i) \neq \alpha(j)$.

The *canonical form* of α is the ω-word obtained by deletion of all redundant letters from α. Two ω-words α, β are *stutter equivalent* if they have the same canonical form. A language is *stutter closed* if it is closed under stutter equivalence.

Theorem 5.1 ([Lam83b]) *Every* $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^0)$ *language is stutter closed.*

Intuitively, it is not very surprising that this principle can be extended to $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ formulae (where $n \in \mathbb{N}_0$). The so-called *n-stuttering* is based on a simple observation that $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ formulae cannot distinguish between $n+1$ and more adjacent occurrences of the same letter in a given ω-word. Formally, a letter $\alpha(i)$ is *n-redundant* if $\alpha(i) = \alpha(i+1) = \ldots = \alpha(i+n+1)$ and there is some $j > i$ such that $\alpha(i) \neq \alpha(j)$. The *n-canonical form*, *n-stutter equivalence*, and *n-stutter closed languages* are defined in the same way as above.

Theorem 5.2 (n-stuttering) *Every* $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ *language is n-stutter closed.*

Proof: The theorem can be proven directly by induction on n. Since it is a consequence of Theorem 5.9, we do not give an explicit proof here[1]. ■

Theorem 5.2 can be used to show that a given property is *not* expressible in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ (or even in LTL). The following corollary is an evidence of this statement.

Corollary 5.3 *The languages* $L_1 = ((aa)^*b)^\omega$ *and* $L_2 = \{aa, ab\}^\omega$ *are not definable in LTL.*

Proof: These languages (already mentioned in Example 3.13) are standard examples of ω-regular languages that are not definable in LTL, i.e. not star-free.

We start with language L_1. For the sake of contradiction, let us assume that there exists a formula $\varphi \in \mathrm{LTL}(\mathsf{U}, \mathsf{X})$ such that $L_1 = L^{\{a,b\}}(\varphi)$. Then $\varphi \in \mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ where $n = \mathsf{U}$-*depth*(φ). Theorem 5.2 implies that L_1 is n-stutter closed. This is a contradiction as words $\alpha = (a^{2n+2}b)^\omega$ and $\beta = (a^{2n+3}b)^\omega$ are n-stutter equivalent and $\alpha \in L_1$ while $\beta \notin L_1$.

The proof for L_2 is analogous. Again, we assume that the language is expressible in LTL. Hence, there is $n \in \mathbb{N}_0$ such that L_2 is expressible in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$. Theorem 5.2 gives us that L_2 is n-stutter closed. This is not true as ω-words $a^{2n+1}ba^\omega \in L_2$ and $a^{2n+2}ba^\omega \notin L_2$ are n-stutter equivalent. ■

Let us recall that the language L_2 can be defined by a formula '$\mathsf{G}_2 a$' with the meaning that a holds at every even position [Wol83].

[1] A direct proof of Theorem 5.2 is of course simpler than the proof of Theorem 5.9. It can be found in [KS04].

5.1.2 Subword stuttering

Since letter stuttering takes into account just the X-*depth* of LTL formulae, a natural question is whether there is another form of stutter-like invariance determined by the U-*depth* of a given LTL formula. We provide a (positive) answer to this question by formulating the principle of *subword stuttering*, which is applicable to $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ formulae (where $m \geq 1$). The term 'subword stuttering' reflects the fact that we do not necessarily delete/pump just individual letters, but whole subwords. The essence of the idea is formulated in the following claim:

Claim 5.4 *Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ where $m \geq 1$. For all $v, u \in \Sigma^*$ and $\alpha \in \Sigma^\omega$ we have that $vu^{m+1}\alpha \models \varphi$ iff $vu^m\alpha \models \varphi$.*

In other words, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ cannot distinguish between m and more adjacent occurrences of the same subword u in a given word. Note that there are no assumptions about the length of u.

Claim 5.4 can be easily proven by induction on m. We just sketch the crucial part of the argument (a full proof is in fact contained in the proof of Theorem 5.9). Let us suppose that $\varphi = \psi \,\mathsf{U}\, \varrho$, where $\psi, \varrho \in \mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^0)$. We want to show that $vu^{m+1}\alpha \models \varphi$ iff $vu^m\alpha \models \varphi$. We concentrate just on the '$\Longrightarrow$' part (the other direction is similar). By induction hypothesis, the following equivalences hold for all $0 \leq \ell < |vu|$:

$$(vu)_\ell \, u^m \, \alpha \models \psi \quad \text{iff} \quad (vu)_\ell \, u^{m-1} \, \alpha \models \psi \tag{5.1}$$

$$(vu)_\ell \, u^m \, \alpha \models \varrho \quad \text{iff} \quad (vu)_\ell \, u^{m-1} \, \alpha \models \varrho \tag{5.2}$$

Let $vu^{m+1}\alpha \models \psi \,\mathsf{U}\, \varrho$. Then there is $j \in \mathbb{N}_0$ such that $(vu^{m+1}\alpha)_j \models \varrho$ and $(vu^{m+1}\alpha)_i \models \psi$ for all $0 \leq i < j$. If $j < |vu|$, we immediately obtain $vu^m\alpha \models \psi \,\mathsf{U}\, \varrho$ by applying (5.1) and (5.2) above. If $j \geq |vu|$, we can imagine that the word $vu^m\alpha$ was obtained from $vu^{m+1}\alpha$ by deletion of the *first* copy of u (from now on, we denote the k^{th} copy of u in $vu^{m+1}\alpha$ by $u[k]$). The situation is illustrated by Figure 5.1. Realize that the (in)validity of ψ and

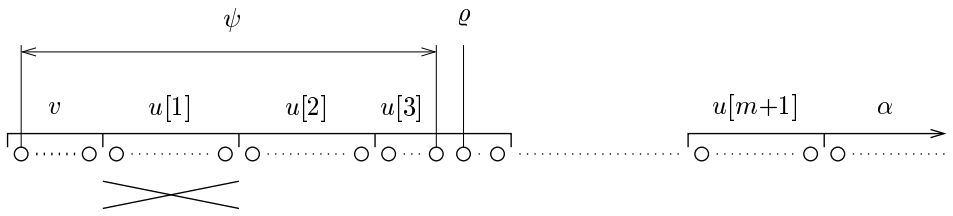

Figure 5.1: Intuition for proof of Claim 5.4.

ϱ for any suffix of $u[2]\,u[3]\,\ldots\,u[m{+}1]\,\alpha$ is not influenced by deletion of the $u[1]$ subword ($\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ contains only future modalities). That is, it

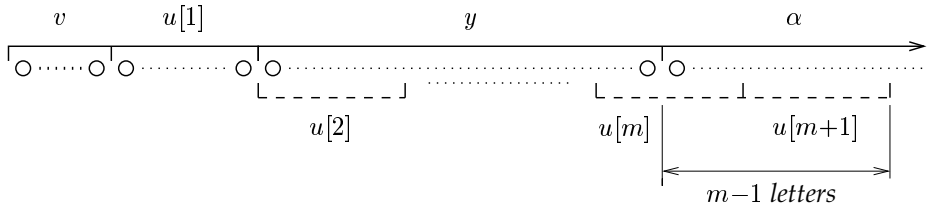

Figure 5.2: Illustration of the first condition of Claim 5.5.

suffices to show that for each suffix v' of v we have that $v'u^{m+1}\alpha \models \psi$ implies $v'u^m\alpha \models \psi$. However, this follows from (5.1) above.

The principle of subword stuttering, as formulated in Claim 5.4, is quite simple and intuitively clear. Now we refine this principle into a stronger form.

Claim 5.5 *Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ where $m \geq 0$. For all $v, y \in \Sigma^*$, $u \in \Sigma^+$, and $\alpha \in \Sigma^\omega$ such that*

- $|y| = |u| \cdot m - m + 1$ *and*
- *y is a prefix of u^ω*

we have that $vuy\alpha \models \varphi$ iff $vy\alpha \models \varphi$.

The structure of $vuy\alpha$ can be illustrated by Figure 5.2. In other words, the u subword has to be repeated 'basically' $m+1$ times as in Claim 5.4, but now we can ignore the last $m-1$ letters of $u[1]\ldots u[m+1]$. Note that there is no assumption about the length of u; if u is 'short' and m is 'large', it can happen that the last $m-1$ letters actually 'subsume' several trailing copies of u.

Claim 5.5 can also be proven by induction on m. Again, we concentrate just on the crucial step when $\varphi = \psi \,\mathsf{U}\, \varrho$ and $\psi, \varrho \in \mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^0)$. We only show the '$\Longrightarrow$' part (the other direction is similar). So, let $vuy\alpha \models \psi \,\mathsf{U}\, \varrho$. Then there is $j \in \mathbb{N}_0$ such that $(vuy\alpha)_j \models \varrho$ and $(vuy\alpha)_i \models \psi$ for all $0 \leq i < j$. We distinguish three possibilities (the first two of them are handled in the same way as in Claim 5.4):

(i) $j < |v|$. To prove that $vy\alpha \models \psi \,\mathsf{U}\, \varrho$, it suffices to show that for every suffix v' of v we have that

- $v'uy\alpha \models \psi$ implies $v'y\alpha \models \psi$,
- $v'uy\alpha \models \varrho$ implies $v'y\alpha \models \varrho$.

However, this follows directly from induction hypothesis.

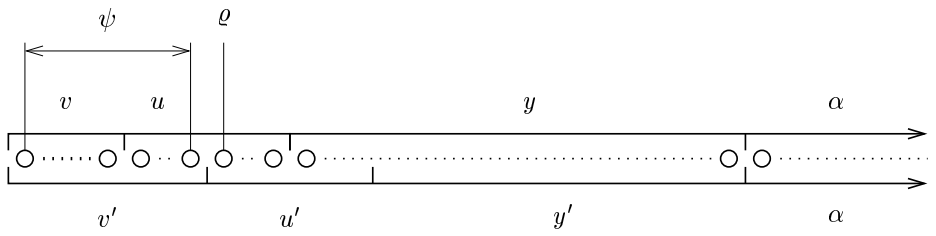

Figure 5.3: Situation in case (iii) of the proof of Claim 5.5.

(ii) $j \geq |vu|$. First, realize that the (in)validity of ψ and ϱ for any suffix of $y\alpha$ is not influenced by deletion of the u subword. Hence, it suffices to show that $v'uy\alpha \models \psi$ implies $v'y\alpha \models \psi$ for each suffix v' of v. This follows from the induction hypothesis in the same way as in (i).

(iii) $|v| \leq j < |vu|$. This requires more care. A key observation is that the word $vuy\alpha$ can be seen as $v'u'y'\alpha = vuy\alpha$, where $|v'| = j$, $|u'| = |u|$, and $|y'| = |y| + |v| - |v'|$. Figure 5.3 depicts the situation. Due to the periodicity of y we have that $vy\alpha = v'y'\alpha$. Hence, it suffices to show that $y'\alpha \models \varrho$ and $v''y'\alpha \models \psi$ for every nonempty suffix v'' of v'. We know that $u'y'\alpha \models \varrho$ and $v''u'y'\alpha \models \psi$; so, if y' is 'sufficiently long', we can use induction hypothesis to finish the proof. That is, we need to verify that $|y'| \geq |u'| \cdot (m-1) - (m-1) + 1$, but this follows immediately from the known (in)equalities $|y'| = |y| + |v| - |v'|$, $|u'| = |u|$, and $|v| > |v'| - |u|$.

5.1.3 General stuttering

In this section we combine the previously discussed principles of letter stuttering and subword stuttering into a single 'general stuttering theorem' which is applicable to $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae.

Definition 5.6 *Let Σ be an alphabet and $m, n \in \mathbb{N}_0$.*

- *A subword $\alpha(i, j)$ of a given $\alpha \in \Sigma^\omega$ is* (m, n)-redundant *if the word $\alpha(i + j, m \cdot j - m + 1 + n)$ is a prefix of $\alpha(i, j)^\omega$.*

- *The relation $\succ_{m,n} \subseteq \Sigma^\omega \times \Sigma^\omega$ is defined as follows: $\alpha \succ_{m,n} \beta$ iff β can be obtained from α by deletion of some (possibly infinitely many) non-overlapping (m, n)-redundant subwords. The* (m, n)-stutter equivalence *is the least equivalence over Σ^ω subsuming the relation $\succ_{m,n}$.*

- *A language $L \subseteq \Sigma^\omega$ is* (m, n)-stutter closed *if it is closed under (m, n)-stutter equivalence.*

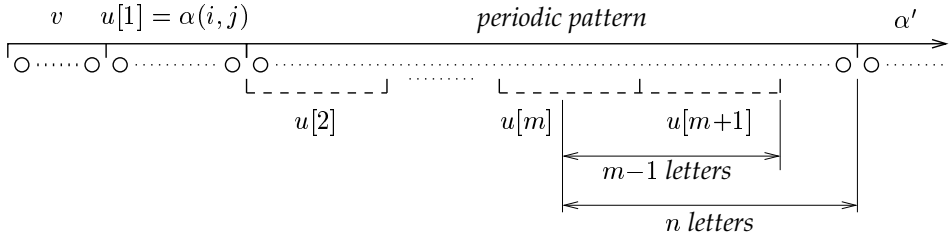

Figure 5.4: Structure of an ω-word with an (m, n)-redundant subword.

The structure of an ω-word α with an (m, n)-redundant subword $\alpha(i, j)$ is illustrated by Figure 5.4. The $\alpha(i, j)$ subword has to be repeated 'basically' $m+1$ times but we can ignore the last $(m-1)-n$ letters (if $(m-1)-n$ is negative, we must actually prolong the repetition 'beyond' the $m + 1$ copies of $\alpha(i, j)$ – see Figure 5.4). Note that there are no assumptions about the sizes of m, n, and j.

Our goal is to prove that the (in)validity of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae is not influenced by deleting/pumping (m, n)-redundant subwords. First, let us realize that this result is a proper generalization of both Theorem 5.2 and Claim 5.5. If we compare the 'periodicity assumptions' of Theorem 5.2, Claim 5.5, and Definition 5.6, we can observe that

- a letter $\alpha(i)$ is n-redundant iff it is consecutively repeated at least $n+1$ times. That is, $\alpha(i)$ is n-redundant iff $\alpha(i + 1, n + 1)$ is a prefix of $\alpha(i, 1)^\omega$. For every $m \in \mathbb{N}_0$ we get that $\alpha(i)$ is n-redundant iff $\alpha(i, 1)$ is (m, n)-redundant as $\alpha(i+1, n+1) = \alpha(i+1, m \cdot 1 - m + 1 + n)$. In other words, the notion of n-redundancy coincides with (m, n)-redundancy for subwords of length 1.

- the condition of Claim 5.5 matches exactly the definition of $(m, 0)$-redundancy.

Before formulating and proving the general stuttering theorem, we need to state two auxiliary lemmata.

Lemma 5.7 *Let Σ be an alphabet, $m, n \in \mathbb{N}_0$, and $\alpha \in \Sigma^\omega$. If a subword $\alpha(i, j)$ is*

(i) (m, n)-redundant then it is also (m', n')-redundant for all $0 \leq m' \leq m$ and $0 \leq n' \leq n$.

(ii) $(m, n + 1)$-redundant then the subword $\alpha(i + 1, j)$ is (m, n)-redundant.

(iii) $(m + 1, n)$-redundant then the subword $\alpha(i + k, j)$ is (m, n)-redundant for every $0 \leq k < j$.

Proof: (i) follows immediately as $j > 0$ implies $m' \cdot j - m' + 1 + n' \leq m \cdot j - m + 1 + n$. (ii) is also simple – due to the $(m, n{+}1)$-redundancy of $\alpha(i, j)$ we know that the subword is repeated at least on the next $m \cdot j - m + 2 + n$ letters. Hence, the subword $\alpha(i{+}1, j)$ is repeated at least on the next $m \cdot j - m + 1 + n$ letters and thus it is (m, n)-redundant. A proof of (iii) is similar; if $\alpha(i, j)$ is repeated on the next $(m{+}1) \cdot j - m + n$ letters, then the subword $\alpha(i{+}k, j)$ (where $0 \leq k < j$) is repeated on the next $(m{+}1) \cdot j - m + n - k = m \cdot j - m + n + j - k$ letters, i.e. $\alpha(i{+}k, j)$ is $(m, n + j - k - 1)$-redundant. The (m, n)-redundancy of $\alpha(i{+}k, j)$ follows from (i) and $k < j$. ■

Lemma 5.8 *For all $m \geq 1$, $n \geq 0$, and all $\alpha, \beta \in \Sigma^\omega$ such that $\alpha \succ_{m,n} \beta$ there exists a surjective function $g : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$ such that*

- (i) *for all $\ell, x \in \mathbb{N}_0$, where $0 \leq x < g(\ell)$, there exists $0 \leq \ell' < \ell$ such that $g(\ell') = x$,*
- (ii) *for each $\ell \in \mathbb{N}_0$ we have that $\alpha_\ell \succ_{m-1,n} \beta_{g(\ell)}$.*

Proof: Let $m \geq 1$, $n \geq 0$ and $\alpha, \beta \in \Sigma^\omega$ such that $\alpha \succ_{m,n} \beta$. Let $D = \alpha(i_0, j_0), \alpha(i_1, j_1), \ldots$ be the (finite or infinite) sequence of non-overlapping (m, n)-redundant subwords which were deleted from α to obtain β (we assume that $i_0 < i_1 < \ldots$). We say that a given $\ell \in \mathbb{N}_0$ is *covered* by a subword $\alpha(i_q, j_q)$ of D if $i_q \leq \ell \leq i_q{+}j_q{-}1$. For each such ℓ we further define $jump(\ell) = \ell + j_q$ and $pos(\ell) = \ell - i_q + 1$. If ℓ is not covered by any subword of D, we put $pos(\ell) = 0$ and $jump(\ell) = \ell$. The set of all ℓ's that are covered by the subwords of D is denoted $cov(D)$. For each $\ell \notin cov(D)$, by $length(\ell)$ we denote the total length of all subwords of D which cover some $k \leq \ell$.

The function g is defined as follows:

$$g(\ell) = \begin{cases} \ell - length(\ell) & \text{if } \ell \notin cov(D), \\ g(jump(\ell)) & \text{otherwise.} \end{cases}$$

Figure 5.5 provides an example of a function g.

In particular, note that uncovered letters of α are projected to the "same" letters in β, and covered letters are in fact mapped to uncovered ones by performing one or more *jumps* of possibly different length. Also note that g is not monotonic in general.

First we show that g is well-defined, i.e. for each $\ell \in cov(D)$ there is $k \in \mathbb{N}$ such that $jump^k(\ell) \notin cov(D)$ (here $jump^k$ denotes $jump$ applied k-times). This is an immediate consequence of the following observation:

For each $\ell \in cov(D)$ there is $k \in \mathbb{N}$ such that $pos(jump^k(\ell)) < pos(\ell)$.

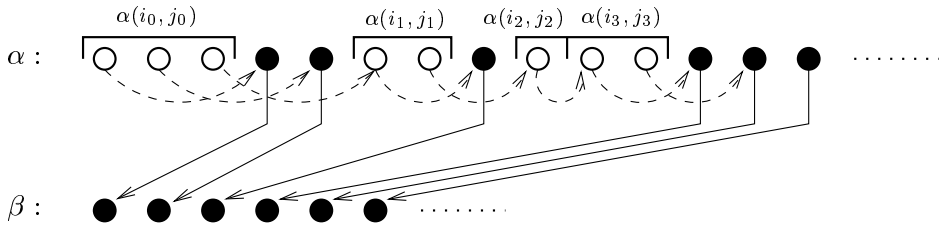

Figure 5.5: An example of a function g constructed in the proof of Lemma 5.8.

Proof of the observation: First, let us realize that $pos(\ell) \geq pos(jump(\ell))$ for every $\ell \in cov(D)$. Now assume that the observation does not hold. Then there is $\ell \in cov(D)$ such that $pos(jump^k(\ell)) = pos(\ell)$ for every $k \in \mathbb{N}$. Let $\alpha(i_q, j_q)$ be the subword of D covering ℓ, and let D_q be the sequence obtained from D by removing the first q elements. Since $pos(jump^k(\ell)) = pos(\ell)$ for every $k \in \mathbb{N}$, all subwords of D_q are adjacent and the length of each of them is at least $pos(\ell)$. Hence, each $\ell' \geq \ell$ is covered by some subword of D_q, which contradicts the assumption that β is infinite.

Proof of (i)*:* First we show that for every $\ell \in \mathbb{N}_0$ we have that $g(\ell + 1) \leq g(\ell) + 1$. Let us assume that there is some $\ell' \in \mathbb{N}_0$ such that $g(\ell' + 1) > g(\ell') + 1$, and let $k \in \mathbb{N}_0$ be the least number such that $\ell = jump^k(\ell')$ is either uncovered or satisfies $g(jump(\ell) + 1) \leq g(jump(\ell)) + 1$. Observe that such a k must exist, and that ℓ satisfies $g(\ell + 1) > g(\ell) + 1$ (otherwise we get a contradiction with the minimality of k). Now we distinguish two possibilities:

- $pos(\ell + 1) \leq 1$. Let ℓ'' be the least uncovered index greater or equal to $\ell + 1$. It follows easily from the definition of g that $g(\ell + 1) = g(\ell'')$. Hence, $g(\ell)$ is either equal to $g(\ell + 1) - 1$ (if $\ell \notin cov(D)$), or greater or equal to $g(\ell + 1)$ (if $\ell \in cov(D)$). Again, this contradicts the assumption that $g(\ell + 1) > g(\ell) + 1$.

- $pos(\ell + 1) \geq 2$. Then $\ell, \ell + 1$ are covered by the same subword of D. By applying the definition of g we obtain $g(\ell) = g(jump(\ell))$ and $g(\ell + 1) = g(jump(\ell+1))$. Moreover, $jump(\ell+1) = jump(\ell)+1$ because $\ell, \ell + 1$ are covered by the same subword of D. If $pos(jump(\ell) + 1)$ equals to 0 or 1, we derive a contradiction using the arguments of previous cases. If $pos(jump(\ell) + 1) \geq 2$, we have that $jump(\ell) \in cov(D)$, hence $g(jump(\ell)+1) \leq g(jump(\ell))+1$ due to the assumption adopted above. Altogether, we derived a contradiction with $g(\ell + 1) > g(\ell) + 1$.

Now we are ready to finish the proof of (i). Let us assume that (i) does not hold, and let $\ell \in \mathbb{N}_0$ be the least number such that (i) is violated for ℓ and some $0 \leq x < g(\ell)$. Clearly $\ell > 0$, because $g(0) = 0$. Further,

$g(\ell - 1) \geq g(\ell) - 1$ due to the claim just proved. This means that either $g(\ell - 1) = x$, or $\ell - 1$ also violates (i). In both cases we have a contradiction with our choice of ℓ.

Proof of (ii)*:* We show that $\alpha_\ell \succ_{m-1,n} \beta_{g(\ell)}$ for each $\ell \in \mathbb{N}_0$. We proceed by induction on $pos(\ell)$.

Basis. $pos(\ell) = 0$. This means that $\ell \notin cov(D)$. Clearly $\alpha_\ell \succ_{m,n} \beta_{g(\ell)}$ because $\beta_{g(\ell)}$ is obtained from α_ℓ by deletion of all those subwords $\alpha(i_q, j_q)$ of D such that $i_q > \ell$. Hence, we also have $\alpha_\ell \succ_{m-1,n} \beta_{g(\ell)}$ by applying Lemma 5.7 (i).

Induction step. Let $pos(\ell) > 0$ and let $k \in \mathbb{N}$ be the least number such that $pos(jump^k(\ell)) < pos(\ell)$. To simplify our notation, we put $\ell' = jump^k(\ell)$. Clearly $g(\ell) = g(\ell')$ by definition of g. By induction hypothesis we have that $\alpha_{\ell'} \succ_{m-1,n} \beta_{g(\ell')}$. Hence, it suffices to show that $\alpha(\ell, \ell' - \ell)$ is a sequence of $(m-1, n)$-redundant subwords. Let us assume that ℓ is covered by $\alpha(i_q, j_q)$. Consider the sequence of subwords

$$\alpha(i_q, j_q), \ldots, \alpha(i_{q+k-1}, j_{q+k-1}).$$

From the minimality of k we obtain that these subwords are adjacent and the length of each of them is at least $pos(\ell)$. Hence, $\alpha(\ell, \ell' - \ell)$ can be seen as a sequence of words

$$\alpha(i_q + pos(\ell) - 1, j_q), \ldots, \alpha(i_{q+k-1} + pos(\ell) - 1, j_{q+k-1}).$$

Moreover, each of these words is $(m-1, n)$-redundant by Lemma 5.7 (iii). ■

Theorem 5.9 (general stuttering) *Every* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ *language is* (m, n)*-stutter closed.*

Proof: Let $m, n \in \mathbb{N}_0$ and $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. It suffices to prove that for all $\alpha, \beta \in \Sigma^\omega$ such that $\alpha \succ_{m,n} \beta$ we have that $\alpha \models \varphi \iff \beta \models \varphi$. We proceed by a simultaneous induction on m and n (we write $(m', n') < (m, n)$ iff $m' \leq m$ and $n' < n$, or $m' < m$ and $n' \leq n$).

Basis. $m = 0$ and $n = 0$. Let $\alpha, \beta \in \Sigma^\omega$ be ω-words such that $\alpha \succ_{0,0} \beta$. Let D denote the sequence of non-overlapping $(0, 0)$-redundant subwords $D = \alpha(i_0, j_0), \alpha(i_1, j_1), \ldots$ which were deleted from α to obtain β (we assume that $i_0 < i_1 < \ldots$). Since $\mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^0)$ formulae are just 'Boolean combinations' of letters and $\top$, it suffices to show that $\alpha(0) = \beta(0)$. If $i_0 > 0$ then it is clearly the case. Now let $i_0 = 0$, and let $k \in \mathbb{N}_0$ be the least number such that the subwords $\alpha(i_k, j_k)$ and $\alpha(i_{k+1}, j_{k+1})$ are not adjacent (i.e. $i_{k+1} > i_k + j_k$). Hence, $\beta(0) = \alpha(i_k + j_k)$ and $(0, 0)$-redundancy of the subwords in D implies that

$$\alpha(0) = \alpha(i_0) = \alpha(i_1) = \alpha(i_2) = \cdots = \alpha(i_k) = \alpha(i_k + j_k) = \beta(0).$$

Induction step. Let $m, n \in \mathbb{N}_0$, and let us assume that the theorem holds for all m', n' such that $(m', n') < (m, n)$. Let $\alpha, \beta \in \Sigma^\omega$ be ω-words such that $\alpha \succ_{m,n} \beta$, and let $D = \alpha(i_0, j_0), \alpha(i_1, j_1), \ldots$ $(i_0 < i_1 < \ldots)$ be the sequence of non-overlapping (m, n)-redundant subwords which were deleted from α to obtain β. We distinguish four possibilities:

- $\varphi \in \mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$ for some $(m', n') < (m, n)$. Since every $\alpha(i, j)$ from D is (m', n')-redundant by Lemma 5.7 (i), we just apply induction hypothesis.
- $\varphi = \mathsf{X}\psi$. We need to prove that $\alpha_1 \models \psi \iff \beta_1 \models \psi$. By induction hypothesis, ψ cannot distinguish between $(m, n-1)$-stutter equivalent ω-words. Hence, it suffices to show that $\alpha_1 \succ_{m,n-1} \beta_1$. If $i_0 > 0$, then $\alpha_1(i_0 - 1, j_0), \alpha_1(i_1 - 1, j_1), \alpha_1(i_2 - 1, j_2), \ldots$ are (m, n)-redundant and due to Lemma 5.7 (i) they are also $(m, n-1)$-redundant. Moreover, β_1 can be obtained from α_1 by deletion of these subwords.

  If $i_0 = 0$, then let $k \in \mathbb{N}_0$ be the least number such that the subwords $\alpha(i_k, j_k)$ and $\alpha(i_{k+1}, j_{k+1})$ are not adjacent. The ω-word β_1 can be obtained from α_1 by deletion of the subwords

  $$\alpha_1(i_0, j_0), \ldots, \alpha_1(i_k, j_k), \alpha_1(i_{k+1} - 1, j_{k+1}), \alpha_1(i_{k+2} - 1, j_{k+2}), \ldots.$$

  The subwords $\alpha_1(i_0, j_0), \alpha_1(i_1, j_1), \ldots, \alpha_1(i_k, j_k)$ are $(m, n-1)$-redundant by Lemma 5.7 (ii), and the other subwords are $(m, n-1)$-redundant by applying Lemma 5.7 (i).
- $\varphi = \psi \mathsf{U} \rho$. By induction hypothesis, ψ, ρ cannot distinguish between $(m-1, n)$-stutter equivalent ω-words. Let g be the function of Lemma 5.8 constructed for the considered m, n, α, β (i.e. $\alpha_\ell \succ_{m-1,n} \beta_{g(\ell)}$ for every $\ell \in \mathbb{N}_0$).

  Now we show that if $\alpha \models \psi \mathsf{U} \rho$ then also $\beta \models \psi \mathsf{U} \rho$. If $\alpha \models \psi \mathsf{U} \rho$, there is $c \geq 0$ such that $\alpha_c \models \rho$ and for every $d < c$ we have that $\alpha_d \models \psi$. By induction hypothesis we get $\beta_{g(c)} \models \rho$. Further, for every $d' < g(c)$ there is $d < c$ such that $g(d) = d'$. By Lemma 5.8, for every $d' < g(c)$ there is $d < c$ such that $\alpha_d \succ_{m-1,n} \beta_{g(d)} = \beta_{d'}$ and hence $\beta_{d'} \models \psi$. Altogether, we obtain that $\beta \models \psi \mathsf{U} \rho$.

  Similarly, we also show that if $\beta \models \psi \mathsf{U} \rho$ then $\alpha \models \psi \mathsf{U} \rho$. If $\beta \models \psi \mathsf{U} \rho$, there is $c \geq 0$ such that $\beta_c \models \rho$ and for every $d < c$ we have that $\beta_d \models \psi$. Let c' be the least number satisfying $g(c') = c$ (there is such a c' because g is surjective). Then $\alpha_{c'} \models \rho$ by induction hypothesis. From the definition of g we get that for every $d' < c'$ it holds that $g(d') < g(c') = c$ (otherwise we would obtain a contradiction with our choice of c'). Thus, $\alpha_{d'} \models \psi$ and hence $\alpha \models \psi \mathsf{U} \rho$.

- φ is a 'Boolean combination' of formulae of the previous cases. Formally, this case is handled by an 'embedded' induction on the structure of φ. The basic step (when φ is *not* of the form $\neg\psi$ or $\psi \wedge \rho$) is covered by the previous cases. The induction step ($\varphi = \neg\psi$ or $\varphi = \psi \wedge \rho$ where we assume that our theorem holds for ψ, ρ) follows immediately. ■

5.2 Stuttering as a sufficient condition

In Section 5.1 we have shown that formulae of certain LTL fragments are invariant under certain forms of stutter equivalence of ω-words. These results (Theorem 5.2, Claim 5.4, Claim 5.5, and Theorem 5.9) were formulated as "pumping lemmata", i.e. necessary conditions which must be satisfied by languages of the respective LTL fragments. In this section we show that certain forms of stutter invariance together with some additional assumptions in fact *characterize* certain LTL fragments.

5.2.1 Letter stuttering

It has been proved by Peled and Wilke [PW97a] that every LTL language closed under stuttering is definable in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^0)$. This proof can be straightforwardly generalized to n-stuttering. Hence, every n-stutter closed LTL property is definable in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$. For the sake of completeness, we present this proof explicitly. (Later we formulate further observations which refer to technical details of this proof.)

Theorem 5.10 *Let $L \subseteq \Sigma^\omega$. The following conditions are equivalent:*

(a) L is definable in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$.

(b) L is definable in LTL and n-stutter closed.

Proof: The $(a) \Longrightarrow (b)$ direction follows from Theorem 5.2. We prove the opposite direction. Given an $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formula φ and $n \in \mathbb{N}_0$, we construct an $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ formula $\tau_n(\varphi)$ and prove that the following observation holds for every alphabet Σ.

$$L^\Sigma(\varphi) \text{ is } n\text{-stutter closed if and only if } L^\Sigma(\varphi) = L^\Sigma(\tau_n(\varphi)).$$

This is sufficient as every LTL language is definable in $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ due to Corollary 3.4.

Let Θ be the set of letters occurring in φ. We set $\theta = \bigvee_{a \in \Theta} p$. Further, for all $a \in \Theta$ and $i > 0$ we define formulae σ_{a^i}, $\sigma_{a^i \neg a}$, $\sigma_{\neg\Theta^i}$, and $\sigma_{\neg\Theta^i\Theta}$ as

follows:

$$\begin{array}{ll} \sigma_{a^1} = a & \sigma_{a^{i+1}} = a \wedge \mathsf{X}\sigma_{a^i} \\ \sigma_{a^0\neg a} = \neg a & \sigma_{a^i\neg a} = a \wedge \mathsf{X}\sigma_{a^{i-1}\neg a} \\ \sigma_{\neg\Theta^1} = \neg\theta & \sigma_{\neg\Theta^{i+1}} = \neg\theta \wedge \mathsf{X}\sigma_{\neg\Theta^i} \\ \sigma_{\neg\Theta^0\Theta} = \theta & \sigma_{\neg\Theta^{i+1}\Theta} = \neg\theta \wedge \mathsf{X}\sigma_{\neg\Theta^i\Theta} \end{array}$$

Observe that

$$\mathsf{X}\text{-}\mathit{depth}(\sigma_{a^{i+1}}) = \mathsf{X}\text{-}\mathit{depth}(\sigma_{a^i\neg a}) = \mathsf{X}\text{-}\mathit{depth}(\sigma_{\neg\Theta^{i+1}}) = \mathsf{X}\text{-}\mathit{depth}(\sigma_{\neg\Theta^i\Theta}) = i.$$

The translation $\tau_n(\varphi)$ is defined inductively on the structure of φ:

- $\tau_n(a) = a$
- $\tau_n(\neg\psi) = \neg\tau_n(\psi)$
- $\tau_n(\psi \wedge \rho) = \tau_n(\psi) \wedge \tau_n(\rho)$
- $\tau_n(\psi \,\mathsf{U}\, \rho) = \tau_n(\psi) \,\mathsf{U}\, \tau_n(\rho)$
- $\tau_n(\mathsf{X}\psi) = \Phi(\psi) \vee \Gamma(\psi)$, where

$$\Phi(\psi) \;=\; (\mathsf{G}\neg\theta \vee \bigvee_{a\in\Theta} \mathsf{G}a) \wedge \tau_n(\psi)$$

and

$$\Gamma(\psi) \;=\; \bigvee_{1\le i\le n+1} (\,\xi(\psi,\neg\Theta,i) \vee \bigvee_{a\in\Theta} \xi(\psi,a,i)\,).$$

The subformulae $\xi(\psi,a,i)$ and $\xi(\psi,\neg\Theta,i)$ of $\Gamma(\psi)$ are constructed as follows:

$$\xi(\psi,a,i) \;=\; \begin{cases} \sigma_{a^i\neg a} \wedge a \,\mathsf{U}\, (\sigma_{a^{i-1}\neg a} \wedge \tau_n(\psi)) & \text{if } i \le n \\ \sigma_{a^{n+1}} \wedge a \,\mathsf{U}\, (\sigma_{a^n\neg a} \wedge \tau_n(\psi)) & \text{if } i = n{+}1 \end{cases}$$

$$\xi(\psi,\neg\Theta,i) \;=\; \begin{cases} \sigma_{\neg\Theta^i\Theta} \wedge \neg\theta \,\mathsf{U}\, (\sigma_{\neg\Theta^{i-1}\Theta} \wedge \tau_n(\psi)) & \text{if } i \le n \\ \sigma_{\neg\Theta^{n+1}} \wedge \neg\theta \,\mathsf{U}\, (\sigma_{\neg\Theta^n\Theta} \wedge \tau_n(\psi)) & \text{if } i = n{+}1 \end{cases}$$

One can readily confirm that the X-*depth*$(\tau_n(\varphi))$ equals n. We need to prove that if $L^\Sigma(\varphi)$ is n-stutter closed, then the languages $L^\Sigma(\varphi)$ and $L^\Sigma(\tau_n(\varphi))$ are the same (the other implication follows directly from Theorem 5.2). Since φ and $\tau_n(\varphi)$ cannot distinguish between letters which do not belong to Θ, we can assume that $\Sigma \subseteq \Theta \cup \{e\}$, where $e \notin \Theta$ represents all letters not occurring in φ.

As both $L^\Sigma(\varphi)$ and $L^\Sigma(\tau_n(\varphi))$ are n-stutter closed (in the latter case we apply Theorem 5.2), it actually suffices to prove that φ and $\tau_n(\varphi)$ cannot be distinguished by any *n-stutter free* ω-word $\alpha \in \Sigma^\omega$ (an ω-word α is n-stutter free if α has no n-redundant letters). That is, for every n-stutter free $\alpha \in \Sigma^\omega$ we show that $\alpha \models \varphi$ iff $\alpha \models \tau_n(\varphi)$. We proceed by induction on the structure of φ. All subcases except for $\varphi = \mathsf{X}\psi$ are trivial. Here we distinguish two possibilities:

- $\alpha = a^\omega$ for some $a \in \Sigma$. Then $\alpha_1 = \alpha$ and thus we get $\alpha \models \mathsf{X}\psi$ iff $\alpha_1 \models \psi$ iff $\alpha_1 \models \tau_n(\psi)$ (by induction hypothesis) iff $\alpha \models \tau_n(\psi)$. Hence, this subcase is 'covered' by the formula $\Phi(\psi)$ saying that α is of the form a^ω and that $\tau_n(\psi)$ holds (the particular case when $\alpha = e^\omega$ corresponds to $\mathsf{G}\neg\theta$).

- $\alpha = a^i b\beta$ where $a, b \in \Sigma$, $a \neq b$, $1 \leq i \leq n+1$, and $\beta \in \Sigma^\omega$.

  Let us first consider the case when $a = e$. Then $a^i b\beta \models \mathsf{X}\psi$ iff $a^{i-1}b\beta \models \psi$ iff $a^{i-1}b\beta \models \tau_n(\psi)$ (we use induction hypothesis). If $i \leq n$, then the last condition is equivalent to $a^i b\beta \models \sigma_{\neg\Theta^i\Theta} \wedge \neg\theta \,\mathsf{U}\, (\sigma_{\neg\Theta^{i-1}\Theta} \wedge \tau_n(\psi))$. If $i = n{+}1$, then the condition is equivalent to $a^{n+1}b\beta \models \sigma_{\neg\Theta^{n+1}} \wedge \neg\theta \,\mathsf{U}\, (\sigma_{\neg\Theta^n\Theta} \wedge \tau_n(\psi))$. In both cases, the resulting formula corresponds to $\xi(\psi, \neg\Theta, i)$.

  The case when $a \in \Theta$ is handled similarly; we have that $a^i b\beta \models \mathsf{X}\psi$ iff $a^{i-1}b\beta \models \psi$ iff $a^{i-1}b\beta \models \tau_n(\psi)$ (by induction hypothesis). If $i \leq n$ then the last condition equals $a^i b\beta \models \sigma_{a^i\neg a} \wedge a \,\mathsf{U}\, (\sigma_{a^{i-1}\neg a} \wedge \tau_n(\psi))$. If $i = n{+}1$ then the condition equals $a^{n+1}b\beta \models \sigma_{a^{n+1}} \wedge a \,\mathsf{U}\, (\sigma_{a^n\neg a} \wedge \tau_n(\psi))$. Anyway, the resulting formula corresponds to $\xi(\psi, a, i)$.

  To sum up, the case when $\alpha = a^i b\beta$ is 'covered' by the formula $\Gamma(\psi)$. ■

In general, the size of $\tau_n(\varphi)$ is exponential in X-*depth*(φ). However, the size of the *circuit*[2] representing $\tau_n(\varphi)$ is only $\mathcal{O}((n+1) \cdot |\varphi|^2)$. To see this, realize the following:

(1) The total size of all circuits representing the formulae $\sigma_{a^n\neg a}, \sigma_{a^{n+1}}$ (for all $a \in \Theta$) and $\sigma_{\neg\Theta^n\Theta}, \sigma_{\neg\Theta^{n+1}}$ is $\mathcal{O}((n+1) \cdot |\varphi|)$. Moreover, all circuits representing the formulae $\sigma_{a^i\neg a}$ and $\sigma_{\neg\Theta^i\Theta}$ (for all $0 \leq i \leq n$) are contained in the circuits representing $\sigma_{a^n\neg a}$ or $\sigma_{\neg\Theta^n\Theta}$, respectively.

(2) Assuming that the circuits of (1) and the circuit representing $\tau_n(\psi)$ are at our disposal, we need to add only a constant number of new nodes to represent the formulae $\xi(\psi, \neg\Theta, i)$ and $\xi(\psi, a, i)$ for given $a \in \Theta$ and $1 \leq i \leq n{+}1$. This means that we need to add $\mathcal{O}((n+1) \cdot |\varphi|)$ new nodes when constructing the circuit for $\tau_n(\mathsf{X}\psi)$.

(3) Since φ contains $\mathcal{O}(|\varphi|)$ subformulae of the form $\mathsf{X}\psi$, the circuit representing φ has $\mathcal{O}((n+1) \cdot |\varphi|^2)$ nodes in total.

Theorem 5.11 *Let φ be an* $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ *formula and $n \in \mathbb{N}_0$. The problem whether there is a formula in* $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ *equivalent to φ is* PSPACE*-complete (assuming unary encoding of n).*

[2] A circuit (or DAG) representing a given LTL formula φ is obtained from the syntax tree of φ by identifying all nodes which correspond to the same subformula.

Proof: The proof employs the fact that validity problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae is PSPACE-complete [SC85].

The observation formulated in proof of Theorem 5.10 directly implies that if there exists an $\mathrm{LTL}(\mathsf{U},\mathsf{X}^n)$ formula equivalent to φ then $\varphi \equiv_i \tau_n(\varphi)$. The circuit representing $\tau_n(\varphi)$ is of size $\mathcal{O}((n+1)\cdot|\varphi|^2)$. The upper bound then follows from the fact that the validity of formula $\varphi \iff \tau_n(\varphi)$ is decidable in PSPACE even if the formula is represented by a circuit [SC85].

The matching lower bound is obtained by reducing the validity problem for $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae (similar reduction of validity problem to problem whether a language is stutter closed appears in [PWW98]). For every $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula ρ we define a formula

$$\pi(\rho) = a \wedge \mathsf{X}a \wedge \mathsf{XX}a \wedge \cdots \wedge \overbrace{\mathsf{XX}\ldots\mathsf{X}}^{n}(a \wedge \mathsf{X}b \wedge \mathsf{XX}\neg\rho).$$

Formula $\pi(\rho)$ is equivalent to an $\mathrm{LTL}(\mathsf{U},\mathsf{X}^n)$ formula if and only if for every alphabet Σ a language $L^\Sigma(\pi(\rho))$ is n-stutter closed. Further, every language defined by formula $\pi(\rho)$ has the form $a^{n+1}bL'$, where L' is a language defined by $\neg\rho$. Therefore, either L' is empty or L is not n-stutter closed. To sum up, $\pi(\rho)$ is equivalent to an $\mathrm{LTL}(\mathsf{U},\mathsf{X}^n)$ formula if and only if $\neg\rho$ is not satisfiable, i.e. if and only if ρ is valid. ■

Theorem 5.10 and the above corollary enable us to extend the statement of Theorem 3.18 to n-stuttering.

Corollary 5.12 *The problem whether an ω-language defined by an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula is n-stutter closed is* PSPACE*-complete (assuming unary encoding of n).*

Proof: Let us consider a language $L = L^\Sigma(\varphi)$, where φ is an $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula. Further, let Θ be the set of letters occurring in φ. There are two cases. First, if $\Theta \subsetneq \Sigma$ then L is n-stutter closed if and only if φ is equivalent to an $\mathrm{LTL}(\mathsf{U},\mathsf{X}^n)$ formula. Second, if $\Theta \supseteq \Sigma$ we set

$$\varphi' = \varphi \wedge \mathsf{G}(\bigvee_{a\in\Sigma} a).$$

Let us note that the size of φ' is linear in the size of φ. As $L = L^{\Sigma'}(\varphi')$ for every $\Sigma' \supseteq \Sigma$, the language L is n-stutter closed if and only if φ' is equivalent to an $\mathrm{LTL}(\mathsf{U},\mathsf{X}^n)$ formula. In both cases the statement is a consequence of Theorem 5.11. ■

The corollary can be alternatively proved by generalization of the techniques developed for (various equivalences including) stutter equivalence in [PWW98].

Finally, let us note that the condition (b) of Theorem 5.10 cannot be weakened to "L is an n-stutter closed ω-regular language", because there

are ω-regular languages which are n-stutter closed for all $n \in \mathbb{N}_0$, yet not definable in LTL. A concrete example of such a language is $L = \{(a^+b^+)^{2^i}c^\omega \mid i \in \mathbb{N}\}$ which is clearly n-stutter closed for every $n \in \mathbb{N}_0$, but not (m,n)-stutter closed for any $m, n \in \mathbb{N}_0$ (and hence not definable in LTL).

5.2.2 General stuttering

In Section 5.2.1 we have shown that $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ languages are exactly n-stutter closed LTL languages. A natural question is whether $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages are fully characterized by the closure property induced by (m,n)-stuttering. In this section we show that this is *not* the case. Nevertheless, regular (m,n)-stutter closed languages are inevitably *noncounting* (see Definition 3.10), and hence expressible in LTL. This means that if L is ω-regular and (m,n)-stutter closed, then $L \in \mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$ for some m', n'. In this section we also show that there is no functional relationship between (m', n') and (m, n).

Theorem 5.13 *Let $L \subseteq \Sigma^\omega$. The following conditions are equivalent:*

(a) L is definable in LTL.

(b) L is ω-regular and noncounting.

(c) L is ω-regular and (m,n)-stutter closed for some $m, n \in \mathbb{N}_0$.

Proof: The equivalence of (a) and (b) is a consequence of several results – see Theorem 3.11 and accompanying comments. The implication (a) $\Longrightarrow$ (c) is given by Theorem 5.9. The implication (c) $\Longrightarrow$ (b) follows from a straightforward observation that a language violating noncounting property is not (m,n)-stutter closed for any $m, n \in \mathbb{N}_0$. ■

A natural question is whether the condition (c) of Theorem 5.13 can be weakened to "L is (m,n)-stutter closed for some $m, n \in \mathbb{N}_0$". The answer is given in our next theorem.

Theorem 5.14 *For all $m \geq 2$ and $n \geq 1$ there is an (m,n)-stutter closed language $L \subseteq \{a,b,c,d\}^\omega$ which is not definable in LTL.*

Proof: Due to Lemma 5.7 (i), we just need to consider the case when $m = 2$ and $n = 1$. We say that a word $w \in \Sigma^*$ is *square-free* if it does not contain a subword of the form uu, where $|u| \geq 1$. It is known that there are infinitely many square-free words[3] $w_0, w_1, \ldots$ over the alphabet $\{a,b,c\}$ [Thu06]. Now observe that for each of these w_i there is no other word

[3]The sequence $w_0, w_1, \ldots$ is defined inductively by $w_0 = a$ and $w_{i+1} = f(w_i)$, where f is a word homomorphism given by $f(a) = abcab$, $f(b) = acabcb$, $f(c) = acbcacb$. The proof in [Thu06] reveals that if w is square-free, then so is $f(w)$.

$v \in \{a, b, c\}^*$ such that $w_i d^\omega \succ_{(2,1)} v d^\omega$ or $v d^\omega \succ_{(2,1)} w_i d^\omega$. This means that $L = \{w_i d^\omega \mid i \in \mathbb{N}_0\}$ is $(2,1)$-stutter closed. Obviously, L is not ω-regular by using standard arguments (pumping lemma for ω-regular languages). Thus, L is not definable in LTL. ■

Due to Theorem 5.13, we know that if L is ω-regular and (m, n)-stutter closed, then L is definable in LTL, i.e. there are $m', n' \in \mathbb{N}$ such that L is definable in $\mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$. However, it is not clear what is the relationship between m, n and m', n'. One might be tempted to think that m', n' can be expressed (or at least bounded) by some simple functions in m, n, for example $m' = m$ and $n' = n$. Our next theorem says that there is no such relationship.

Theorem 5.15 *Let $m \geq 2$ and $n \geq 1$. For all $m', n' \in \mathbb{N}_0$ there is an (m, n)-stutter closed LTL language $L \subseteq \{a, b, c, d\}^\omega$ which is not definable in $\mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$.*

Proof: First, realize that for all $m', n' \in \mathbb{N}_0$ there are only finitely many pairwise non-equivalent $\mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$ formulae over the alphabet $\{a, b, c, d\}$. Hence, it suffices to show that for all $m \geq 2$ and $n \geq 1$ there are infinitely many (m, n)-stutter closed LTL languages over the alphabet $\{a, b, c, d\}$. Due to Lemma 5.7 (i), we just need to consider the case when $m = 2$ and $n = 1$. Let L be the language constructed in the proof of Theorem 5.14. Now realize that each of the infinitely many finite subsets of L is a $(2, 1)$-stutter closed LTL language. ■

Finally, let us note that possible generalizations of Theorem 5.14 and Theorem 5.15 cannot cross certain limits – they do not hold for all $m, n \in \mathbb{N}_0$ and every alphabet Σ. For example, every $(1, 0)$-stutter closed language over the alphabet $\{a, b\}$ is definable in $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$. To see this, realize that the quotient of $\{a, b\}^\omega$ under $(1, 0)$-stutter equivalence has exactly eight equivalence classes represented by words $(ab)^\omega$, $(ba)^\omega$, a^ω, b^ω, ab^ω, ba^ω, aba^ω, and bab^ω. Hence, there are exactly $2^8 = 256$ languages over $\{a, b\}$ which are $(1, 0)$-stutter closed. Since each equivalence class of the quotient is a language definable in $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$, we can conclude that each of these 256 languages is definable in $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$.

5.3 Answers to Questions 1, 2, and 3

Now we are ready to provide answers to Questions 1, 2, and 3 which were stated at the beginning of this chapter (though Question 3 will be left open in fact). We start with a simple observation.

Lemma 5.16 *For each $n \geq 1$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^n)$ which cannot be expressed in $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^{n-1})$.*

Proof: Let $\Sigma = \{a, b\}$ and $n \geq 1$. Consider the formula $\varphi \equiv \overbrace{\mathsf{X}\mathsf{X}\ldots\mathsf{X}}^{n} a$. We show that $L^{\Sigma}(\varphi)$ is not closed under $(n{-}1)$-stutter equivalence (which suffices due to Theorem 5.2). It is easy; realize that $a^{n+1}b^{\omega} \in L^{\Sigma}(\varphi)$ and the first occurrence of a in this word is $(n{-}1)$-redundant. Since $a^{n}b^{\omega} \notin L^{\Sigma}(\varphi)$, we are done. ■

A 'dual' fact is proven below (it is already non-trivial).

Lemma 5.17 *For each $m \geq 1$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ which cannot be expressed in $\mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X})$.*

Proof: Let $m \geq 1$ and let $\Sigma = \{b, a_1, \ldots, a_m\}$. We define a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ as

$$\varphi = \mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \cdots \wedge \mathsf{F}(a_{m-1} \wedge \mathsf{F}a_m) \ldots)).$$

Let us fix an arbitrary $n \in \mathbb{N}_0$, and define a word $\alpha \in \Sigma^{\omega}$ by

$$\alpha = (b^{n+1}\, a_m\, a_{m-1} \ldots a_1)^m\, b^{\omega}.$$

Clearly $\alpha \models \varphi$ and the subword $\alpha(0, n{+}1{+}m)$ is $(m{-}1, n)$-redundant. As the word β obtained from α by removing $\alpha(0, n{+}1{+}m)$ does not model φ, the language $L^{\Sigma}(\varphi)$ is not $(m{-}1, n)$-stutter closed. As it holds for every $n \in \mathbb{N}_0$, the formula φ is not expressible in $\mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X})$. ■

The last technical lemma which is needed to formulate answers to Questions 1 and 2 follows.

Lemma 5.18 *For all $m, n \in \mathbb{N}_0$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ which is expressible neither in $\mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^n)$ (assuming $m \geq 1$), nor in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n-1})$ (assuming $n \geq 1$).*

Proof: If $m = 0$ or $n = 0$, we can apply Lemma 5.16 or Lemma 5.17, respectively. Now let $m, n \geq 1$, and let $\Sigma = \{a_1, \ldots, a_k, b\}$ where $k = \max\{m, n{+}1\}$. We define formulae ψ and φ as follows:

$$\psi \;=\; \begin{cases} a_m \wedge \mathsf{X}^n a_{m-n} & \text{if } m > n \\ a_m \wedge \mathsf{X}^n a_{m+1} & \text{if } m \leq n \end{cases}$$

$$\varphi \;=\; \begin{cases} \mathsf{F}\psi & \text{if } m = 1 \\ \mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \mathsf{F}(a_3 \wedge \cdots \wedge \mathsf{F}(a_{m-1} \wedge \mathsf{F}\psi) \ldots))) & \text{if } m > 1 \end{cases}$$

where X^l abbreviates $\overbrace{\mathsf{X}\mathsf{X}\ldots\mathsf{X}}^{l}$. The formula φ belongs to $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. Let us consider the ω-word α defined as

$$\alpha = \begin{cases} (a_m\, a_{m-1} \ldots a_1)^m a_m\, a_{m-1} \ldots a_{m-n+1} q^{\omega} & \text{if } m > n \\ (a_{n+1}\, a_n \ldots a_1)^{m+1} q^{\omega} & \text{if } m = n \\ (a_{n+1}\, a_n \ldots a_1)^{m+1} a_{n+1}\, a_n \ldots a_{m+2} q^{\omega} & \text{if } m < n \end{cases}$$

It is easy to check that $\alpha \in L^{\Sigma}(\varphi)$ and that the subword $\alpha(0,k)$ (where $k = \max\{m, n+1\}$) is $(m, n-1)$-redundant as well as $(m-1, n)$-redundant. As the word β obtained from α by removing $\alpha(0,k)$ does not satisfy φ, the language $L^{\Sigma}(\varphi)$ is neither $(m, n-1)$-stutter closed, nor $(m-1, n)$-stutter closed. ■

The knowledge presented in the three lemmata above allows to conclude the following:

Corollary 5.19 (Answer to Question 1) *The* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, *and* $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ *hierarchies are strict.*

Corollary 5.20 (Answer to Question 2) *Let A and B be classes of* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, *or* $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ *hierarchy (not necessarily of the same one) such that A is syntactically not included in B. Then there is a formula $\varphi \in A$ which cannot be expressed in B.*

Although we cannot provide a satisfactory answer to Question 3, we can at least reject the aforementioned 'natural' hypotheses (see the beginning of this chapter).

Lemma 5.21 (About Question 3) *For all $m, n \in \mathbb{N}_0$ there is a language definable in* $\mathrm{LTL}(\mathsf{U}^{m+2}, \mathsf{X}^n)$ *as well as in* $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^{n+1})$ *which is not definable in* $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$.

Proof: We start with the case when $m = n = 0$. Let $\Sigma \supseteq \{a, b\}$, and let $\psi_1 = \mathsf{F}(b \wedge (b \,\mathsf{U}\, \neg b))$ and $\psi_2 = \mathsf{F}(b \wedge \mathsf{X}\neg b)$. Note that $\psi_1 \in \mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ and $\psi_2 \in \mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^1)$. Moreover, ψ_1 and ψ_2 are equivalent as they define the same language $L = \Sigma^* b (\Sigma \setminus \{b\}) \Sigma^\omega$. This language is not definable in $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ as it is not $(1,0)$-stutter closed; for example, the ω-word $\alpha = abab^\omega \in L$ contains a $(1,0)$-redundant subword $\alpha(0,2)$ but $\alpha_2 = ab^\omega \notin L$.

The above example can be generalized to arbitrary m, n (using the designed formulae ψ_1, ψ_2). For given m, n we define formulae $\varphi_1 \in \mathrm{LTL}(\mathsf{U}^{m+2}, \mathsf{X}^n)$ and $\varphi_2 \in \mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^{n+1})$, both defining the same language L over $\Sigma = \{b, a, a_1, \ldots, a_{m+1}\}$, and we give an example of an ω-word $\alpha \in L$ with an $(m+1, n)$-redundant subword such that α without this subword is not from L. We distinguish three cases.

- $m = n > 0$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{XF}(a \wedge \mathsf{XF}(a \wedge \mathsf{XF}(a \wedge \cdots \wedge \mathsf{XF}(a \wedge}^{m\text{-times}} \psi_i) \ldots))).$$

  The ω-word $\alpha = (ab)^{m+2} b^\omega \in L$, $\alpha(0,2)$ is $(m+1, n)$-redundant, and $\alpha_2 = (ab)^{m+1} b^\omega \notin L$.

- $m > n$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{XF}(b \wedge \mathsf{XF}(b \wedge \cdots \wedge \mathsf{XF}(b \wedge}^{n\text{-times}} \varphi_i') \ldots)),$$

where

$$\varphi_i' = \overbrace{\mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \cdots \wedge \mathsf{F}(a_{m-n} \wedge}^{(m-n)\text{-times}} \psi_i) \ldots)).$$

The ω-word $\alpha = (ba_{m-n}a_{m-n-1}\ldots a_1)^{m+1}b^\omega \in L$, $\alpha(0, m-n+1)$ is $(m+1, n)$-redundant, and $\alpha_{m-n+1} \notin L$.

- $m < n$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \cdots \wedge \mathsf{F}(a_m \wedge}^{m\text{-times}} \overbrace{\mathsf{XX}\ldots\mathsf{X}}^{n} \psi_i) \ldots)).$$

The ω-word $\alpha = (b^{n-m}a_{m+1}a_m \ldots a_1)^{m+2}b^\omega \in L$, $\alpha(0, n+1)$ is $(m+1, n)$-redundant, and $\alpha_{n+1} \notin L$. ■

In fact, the previous lemma says that if we take two classes $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ which are syntactically incomparable and where $m_1, m_2 \geq 1$, then their semantic intersection (i.e. the intersection of the corresponding classes of languages) is strictly greater than the class of languages definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ where $m = \min\{m_1, m_2\}$ and $n = \min\{n_1, n_2\}$. Another consequence of Lemma 5.21 is that there is generally no "best" way how to minimize the nesting depths of X and U modalities in a given LTL formula.

5.4 Application in model checking

In this section we indicate that letter stuttering and general stuttering principles can potentially improve partial order reduction methods (see Subsection 2.4.1).

5.4.1 Letter stuttering

Most of the partial order reduction methods designed so far are based on stutter equivalence. Therefore, they can be used only for verification of stutter closed properties. All such properties are definable in $\mathrm{LTL}(\mathsf{U})$. This fact contributes to the current situation where many people working in model checking believe that it is natural to write specification formulae without X modality (this opinion had been originally formulated by Lamport [Lam83b] long time before the first partial order reduction algorithm has been suggested). In spite of this, the use of X operator can bring some benefits.

- Some properties that are stutter closed can be defined by a $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formula in a more readable way. Many patterns of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae defining stutter closed properties can be found in [PC03].

- There are some systems (e.g. synchronous hardware) where one transition step has a clear meaning and the study of relations between successive states makes good sense. In context of such systems, it is obvious that one wants to write and verify some specification formulae employing X operator.[4] If we know that the formula defines a stutter closed property (this can be checked in PSPACE due to Theorem 3.18), we can use a combination of automata-based model checking algorithm and classic partial order reduction method. However, these partial order reduction methods cannot be employed if the property is not stutter closed. Here is a potential application of n-stuttering.

Let $\varphi \in \mathrm{LTL}(\mathsf{U},\mathsf{X})$ be a specification formula and $n = \mathsf{X}\text{-}\mathit{depth}(\varphi)$. We know that φ cannot distinguish between ω-words that are n-stutter equivalent. This observation can be seen as the first step towards new partial order reduction methods (see Subsection 2.4.1). Unfortunately, we do not have any reduction algorithm based on the n-stuttering equivalences yet.

It is worth mentioning that increase of $n = \mathsf{X}\text{-}\mathit{depth}(\varphi)$ implies less effective reduction (measured by the size of reduced system). This accords with the fact that for every n there exist words which cannot be distinguished by any $\mathrm{LTL}(\mathsf{U},\mathsf{X}^{n+1})$ formula but they can be distinguished by a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X}^{n+1})$.

5.4.2 General stuttering

Motivation for development of partial order reduction methods based on general stuttering is straightforward as these methods have a potential to improve actual partial order reduction algorithms designed for $\mathrm{LTL}(\mathsf{U})$ formulae. An evidence is provided by the following example. Moreover, general stuttering is not limited to X-free formulae.

Example 5.22 *Let us consider the Kripke structure defined in Example 2.25 and depicted in Figure 2.3. Further, assume that we want to check whether the system satisfies a specification formula $\varphi \in \mathrm{LTL}(\mathsf{U}^1)$ with atomic propositions dependent just on the value of variable x like, for example, $\varphi = \mathsf{G}(\mathrm{x} < 8)$. In this case, it is sufficient to check the reduced structure given by Figure 5.6 as every (word over $2^{At(\varphi)}$ corresponding to some) path in the original structure is $(1,0)$-stutter equivalent to a (word over $2^{At(\varphi)}$ corresponding to some) path in the reduced structure*

[4] We have to say that if one wants to verify systems like synchronous hardware, symbolic model checking algorithms are usually the best choice.

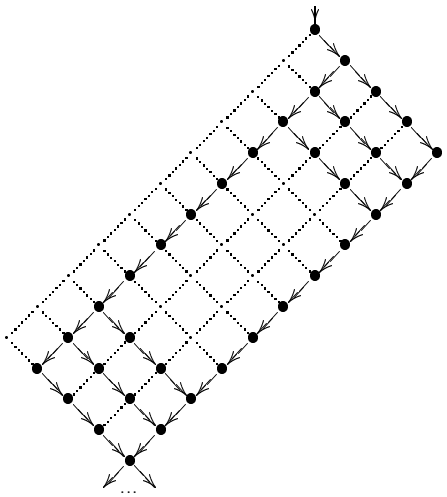

Figure 5.6: The Kripke structure reduced with use of general stuttering.

and vice versa. The original Kripke structure reduced by standard partial order reduction (based on stutter equivalence) is depicted in Figure 2.4.

As in the case of letter stuttering, any reduction algorithm based on general stuttering has not been suggested yet. The reduced structure in the example above is made by hand.

As in the case of letter stuttering, the effectiveness of reduction based on (m, n)-stutter equivalence decreases with increase of m and/or n.

5.5 Additional notes

Although the results presented in this chapter are formulated in terms of infinite words, all of them carry over to finite words. Moreover, some of the definitions and proofs working with finite words are simpler. For example, a letter of a finite word is n-redundant if it is immediately followed by $n + 1$ copies of the same letter (without other requirements). Further, the (m, n)-stutter equivalence can be defined as the least equivalence over Σ^* such that if a word u has an (m, n)-redundant subword then u is equivalent to u without the subword. This simpler definition allows to prove the general stuttering theorem for languages of finite words in a shorter and more transparent way. For details on general stuttering over finite words we refer to the paper [KS02] written in this setting.

We also note that the construction of a formula $\tau_n(\varphi)$ in the proof of Theorem 5.10 is slightly different when we work with LTL defined over atomic propositions instead of letters. This modified construction of $\tau_n(\varphi)$

can be found in [KS04] together with all results concerning n-stuttering (including precise proof of Theorem 5.2).

The chapter encompasses two questions that were not answered. Question 3 was already discussed at the beginning of this chapter. A much more important question would be whether the new stuttering principles can help to improve current state space reduction methods or model checking algorithms in general. A prominent goal for our future research is to create a reduction method or verification algorithm demonstrating that the answer is positive.

Another topic for future work is to extend the presented stuttering principles to LTL with past modalities.

Chapter 6

Characteristic patterns

The chapter is based on the results presented in [KS05a].

In this chapter we introduce a new concept of *characteristic patterns*[1]. Roughly speaking, for each alphabet Σ and all $m, n \in \mathbb{N}_0$ we design a finite set of (m, n)-*patterns*, where each (m, n)-pattern is a finite object representing an ω-language over Σ so that the following conditions are satisfied:

- Each $\alpha \in \Sigma^\omega$ is represented by exactly one (m, n)-pattern (consequently, the sets of ω-words represented by different patterns are disjoint).
- ω-words which are represented by the same (m, n)-pattern cannot be distinguished by any formula of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.
- For each (m, n)-pattern p we can effectively construct a formula $\psi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ so that for each $\alpha \in \Sigma^\omega$ it holds that $\alpha \models \psi$ if and only if α is represented by the pattern p.

Thus, the semantics of each formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ is fully characterized by a finite subset of (m, n)-patterns, and vice versa. Intuitively, the (m, n)-patterns represent exactly the information about ω-words which determines the (in)validity of all $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae. The patterns are defined inductively on m, and the inductive step brings some insight into what is actually gained (i.e. what new properties can be expressed) by increasing the nesting depth of U by one. We refer to Section 6.1 for further remarks which aim at providing better intuition behind (m, n)-patterns. In the same section we also give proper definition, basic theorems about patterns, and an evidence that characteristic patterns can be used as a tool for proving further results about the logic LTL and its fragments. In particular, they can be used to construct a short proof of a (somewhat simplified)

[1] Let us note that these patterns have nothing to do with the forbidden patterns defined in Subsection 3.2.2.

form of stutter invariance of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ presented in the previous chapter. This, in turn, allows to construct simpler proofs of strictness of the $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchies.

In Section 6.2 we provide an algorithm for model checking of patterns and identify three potential applications of characteristics patterns in model checking area. For didactic reasons, we work with existential model checking problem in the first two cases. However, all ideas can be modified for universal model checking in a straightforward way.

1. Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula specifying the property to be verified. Characteristic patterns can be used to decompose the formula into an equivalent disjunction $\psi_1 \vee \ldots \vee \psi_k$ of mutually exclusive formulae (i.e. we have $\psi_i \Rightarrow \bigwedge_{j \neq i} \neg\psi_j$ for each i). Roughly speaking, each ψ_i corresponds to one of the patterns which define the semantics of φ. Hence, the ψ_i formulae are not necessarily smaller or simpler than φ from the syntactic point of view. The simplification is on semantic level, because each ψ_i "cuts off" a dedicated subset of runs that satisfy φ. Another advantage of this method is its scalability – the patterns can be constructed also for those n and m that are larger than the nesting depths of X and U in φ. Thus, the patterns can be repeatedly "refined", which corresponds to decomposing the constructed ψ_i formulae. Another way how to refine the patterns is to enlarge the alphabet Σ. This is indeed sensible, because in this way we can further split the sets of runs of a given system into more manageable subsets.

   This decomposition technique enables the following model checking strategy: First try to model-check φ. If this does not work (because of, for example, memory overflow), then decompose φ into $\psi_1 \vee \ldots \vee \psi_k$ and try to model-check the $\psi_1, \ldots, \psi_k$ formulae. This can be done sequentially or even in parallel. If at least one subtask produces a positive answer, we are done (there is a run satisfying φ). Similarly, if all subtasks produce a negative answer, we are also done (there is no such run). Otherwise, we go on and decompose those ψ_i for which our model checker did not manage to answer.

   Obviously, the introduced strategy can only lead to better results than checking just φ, and it is completely independent of the underlying model checker. Moreover, some new and relevant information is obtained even in those cases when this strategy does not lead to a definite answer – we know that if there is a run satisfying φ, it must satisfy some of the subformulae we did not manage to model check. The level of practical usability of the above discussed approach can only be measured by outcomes of practical experiments which are

beyond the scope of this work.[2] For a more detailed explanation of the decomposition technique and a discussion of its potential benefits and drawbacks see Subsection 6.2.1.

Similar decomposition techniques for universal model checking have been proposed in [McM99] and [Zha03]. In [McM99], a specification formula of the form $\mathsf{G}\varphi$ is decomposed into a set of formulae

$$\{\mathsf{G}(x{=}v_i \Rightarrow \varphi) \mid v_i \text{ is in the range of the variable } x\}.$$

A system satisfies $\mathsf{G}\varphi$ if and only if it satisfies all formulae of the set. This decomposition technique has been implemented in the SMV system together with methods aimed at reducing the range of x. This approach has then been used for verification of specific types of infinite-state systems (see [McM99] for more details). In [Zha03], a given specification formula φ is model-checked as follows: First, a finite set of formulae $\psi_1, \ldots, \psi_n$ of the form $\psi_i = \mathsf{G}(x{\neq}v_0 \Rightarrow x{=}v_i)$ is constructed such that the verified system satisfies $\psi_1 \vee \ldots \vee \psi_n$. The formulae $\psi_1, \ldots, \psi_n$ are either given directly by the user, or constructed automatically using methods of static analysis. The system satisfies φ if and only if it satisfies a formula $\psi_i \Rightarrow \varphi$ for all i. Using this approach, the peak memory in model checking has been reduced by 13–25% in the three case studies included in the paper.

2. Characteristic patterns could be potentially used also in a different way: instead of checking whether a given system exhibits a run corresponding to a given pattern (this is what we do above), we could first extract all the patterns that can be exhibited by the system, and then check whether there is one for which φ holds. This makes sense in situations when we want to check a large number of formulae on the same system. The patterns fully characterize the system's behaviour (with respect to properties expressible in a given $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ fragment), and this information could be re-used when checking the individual formulae. Unfortunately, the set of all patterns exhibited by a given system seems to be computable only in restricted cases. In Subsection 6.2.2 we present an algorithm for model checking a path based on characteristic patterns.

3. Subsection 6.2.3 indicates potential improvement of partial order reduction methods with use of characteristic patterns.

Section 6.3 contains additional notes about patterns including one open question.

[2] Practical implementation of the method is under preparation.

6.1 Definitions and basic theorems

To get some intuition about characteristic patterns, let us first consider the set of patterns constructed for the alphabet $\Sigma = \{a, b, c\}$, $m = 1$, and $n = 0$ (as we shall see, the m and n correspond to the nesting depths of U and X, respectively). Let $\alpha \in \Sigma^\omega$ be an ω-word. A letter $\alpha(i)$ is *repeated* if there is $j < i$ such that $\alpha(j) = \alpha(i)$. The $(1,0)$-pattern of α, denoted $pat(1, 0, \alpha)$, is the finite word obtained from α by deletion of all repeated letters (for reasons of consistent notation, this word is written in parenthesis). For example, if $\alpha = \underline{a}a\underline{b}bbaababab\underline{c}abccacab\ldots$, then $pat(1, 0, \alpha) = (abc)$. So, the set of all $(1, 0)$-patterns over the alphabet $\{a, b, c\}$, denoted $Pats(1, 0, \{a, b, c\})$, has exactly 15 elements which are the following:

$$(abc), (acb), (bac), (bca), (cab), (cba), (ab), (ba), (ac), (ca), (bc), (cb), (a), (b), (c)$$

Thus, the set $\{a, b, c\}^\omega$ is divided into 15 disjoint subsets, where each set consists of all ω-words that have a given pattern. It remains to explain why these patterns are interesting. The point is that $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formulae can actually express just the order of non-repeated letters. For example, the formula $a \,\mathsf{U}\, b$ says that either the first non-repeated letter is b, or the first non-repeated letter is a and the second one is b. So, this formula holds for a given $\alpha \in \{a, b, c\}^\omega$ if and only if $pat(1, 0, \alpha)$ equals to

$$(b), (ba), (bc), (bac), (bca), (ab), \text{ or } (abc).$$

We claim (and later also prove) that ω-words of $\{a, b, c\}^\omega$ which have the same $(1, 0)$-pattern cannot be distinguished by any $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formula. So, a language defined by a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ over alphabet $\Sigma = \{a, b, c\}$ is fully characterized by a subset of $Pats(1, 0, \{a, b, c\})$. Moreover, for each $p \in Pats(1, 0, \{a, b, c\})$ we can construct an $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formula φ_p such that for every $\alpha \in \{a, b, c\}^\omega$ we have that $\alpha \models \varphi_p$ iff $pat(1, 0, \alpha) = p$. For example,

$$\varphi_{(abc)} = a \ \wedge\ (a \,\mathsf{U}\, b) \ \wedge\ ((a \vee b) \,\mathsf{U}\, c).$$

To indicate how this can be generalized to larger m and n, we show how to extract a $(2, 0)$-pattern from a given $\alpha \in \{a, b, c\}^\omega$. We start by considering an infinite word over the alphabet $Pats(1, 0, \{a, b, c\})$ constructed as follows:

$$pat(1, 0, \alpha_0)\ pat(1, 0, \alpha_1)\ pat(1, 0, \alpha_2)\ pat(1, 0, \alpha_3)\ldots$$

For example, for $\alpha = aabaca^\omega$ we get the sequence

$$(abc)(abc)(bac)(ac)(ca)(a)^\omega.$$

The pattern $pat(2, 0, \alpha)$ is obtained from the above sequence by deletion of repeated letters (realize that now we consider the alphabet

$Pats(1, 0, \{a, b, c\})$). Hence,

$$pat(2, 0, \alpha) = ((abc)(bac)(ac)(ca)(a)).$$

Similarly as above, it holds that those ω-words of $\{a, b, c\}^\omega$ which have the same $(2, 0)$-pattern cannot be distinguished by any $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ formula. Moreover, for each $p \in Pats(2, 0, \{a, b, c\})$ there is an $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ formula φ_p such that for every $\alpha \in \{a, b, c\}^\omega$ we have that $\alpha \models \varphi_p$ iff $pat(2, 0, \alpha) = p$.

Formally, we consider every finite sequence of $(1, 0)$-patterns, where no $(1, 0)$-pattern is repeated, to be a $(2, 0)$-pattern. This makes the inductive definition simpler, but in this way we also introduce patterns that are not "satisfiable". For example, there is obviously no $\alpha \in \{a, b, c\}^\omega$ such that $pat(2, 0, \alpha) = ((a)(ab))$.

The last problem we have yet not addressed is how to deal with the X operator. First note that the X operator can be pushed towards letters using the following equivalences (see, for example, [Eme90]):

$$\mathsf{X}\top \Leftrightarrow \top \qquad \mathsf{X}(\neg\varphi) \Leftrightarrow \neg\mathsf{X}\varphi$$
$$\mathsf{X}(\varphi_1 \,\mathsf{U}\, \varphi_2) \Leftrightarrow \mathsf{X}\varphi_1 \,\mathsf{U}\, \mathsf{X}\varphi_2 \qquad \mathsf{X}(\varphi_1 \wedge \varphi_2) \Leftrightarrow \mathsf{X}\varphi_1 \wedge \mathsf{X}\varphi_2$$

Note that the nesting depth of X remains unchanged by performing this transformation. Hence, we can safely assume that the X operator occurs in LTL formulae only within subformulae of the form $\mathsf{X}\mathsf{X}\ldots\mathsf{X}a$. This is the reason why we can handle the X operator in the following way: the set $Pats(m, n, \Sigma)$ is defined in the same way as $Pats(m, 0, \Sigma)$. The only difference is that we start with the alphabet Σ^{n+1} instead of Σ.

Now we present a full formal definition of characteristic patterns.

Definition 6.1 *Let Σ be an alphabet. For all $m, n \in \mathbb{N}_0$ we define the set $Pats(m, n, \Sigma)$ inductively as follows:*

- $Pats(0, n, \Sigma) = \{w \in \Sigma^* \mid |w| = n+1\}$
- $Pats(m+1, n, \Sigma) = \{(p_1 \ldots p_k) \mid k \in \mathbb{N},\ p_1, \ldots, p_k \in Pats(m, n, \Sigma),$ $p_i \neq p_j$ *for* $i \neq j\}$

The size of $Pats(m, n, \Sigma)$ and the size of its elements are estimated in our next lemma.

Lemma 6.2 *For every $i \in \mathbb{N}_0$, let us define the function $fac_i : \mathbb{N}_0 \to \mathbb{N}_0$ inductively as follows:*

$$fac_i(x) = \begin{cases} x & \text{if } i = 0 \\ (fac_{i-i}(x) + 1)! & \text{otherwise} \end{cases}$$

The number of elements of $Pats(m, n, \Sigma)$ is bounded by $fac_m(|\Sigma|^{n+1})$, and the size of each $p \in Pats(m, n, \Sigma)$ is bounded by $(n + 1) \cdot \Pi_{i=0}^{m-1} fac_i(|\Sigma|^{n+1})$.

Proof: Directly from definitions. ■

The bounds given in Lemma 6.2 are non-elementary in m. This indicates that all of our algorithms are computationally unfeasible from the asymptotic analysis point of view. However, LTL formulae that are used in practice typically have a small nesting depth of U (usually not larger than 3 or 4), and do not contain any X operators. In this light, the bounds of Lemma 6.2 can be actually interpreted as "good news", because even a relatively small formula φ can be decomposed into a disjunction of many formulae which refine the meaning of φ.

To all $m, n \in \mathbb{N}_0$ and $\alpha \in \Sigma^\omega$ we associate a unique pattern of $Pats(m, n, \Sigma)$. This definition is again inductive.

Definition 6.3 *Let $\alpha \in \Sigma^\omega$. For all $m, n \in \mathbb{N}_0$ we define the* characteristic (m, n)-pattern *of α, denoted $pat(m, n, \alpha)$, and* (m, n)-pattern word *of α, denoted $patword(m, n, \alpha)$, inductively as follows:*

- $pat(0, n, \alpha) = \alpha(0) \dots \alpha(n)$,
- $patword(m, n, \alpha) \in Pats(m, n, \Sigma)^\omega$ *is defined by* $patword(m, n, \alpha)(i) = pat(m, n, \alpha_i)$,
- $pat(m{+}1, n, \alpha)$ *is the finite word (written in parenthesis) obtained from $patword(m, n, \alpha)$ by deletion of all repeated letters.*

Words $\alpha, \beta \in \Sigma^\omega$ are said to be (m, n)-pattern equivalent, *written $\alpha \sim_{m,n} \beta$, if $pat(m, n, \alpha) = pat(m, n, \beta)$.*

Example 6.4 *Let us consider an ω-word $\alpha = abbbacbac(ba)^\omega$. Some examples of characteristic patterns follow. Underlined sequences correspond to $(0, n)$-patterns, where $n > 0$.*

$$
\begin{aligned}
pat(0,0,\alpha) &= a \\
patword(0,0,\alpha) &= abbbacbac(ba)^\omega = \alpha \\
pat(1,0,\alpha) &= (abc) \\
patword(1,0,\alpha) &= (abc)(bac)(bac)(bac)(acb)(cba)(bac)(acb)(cba)((ba)(ab))^\omega \\
pat(2,0,\alpha) &= ((abc)(bac)(acb)(cba)(ba)(ab)) \\
pat(0,1,\alpha) &= \underline{ab} \\
patword(0,1,\alpha) &= \underline{ab}\,\underline{bb}\,\underline{bb}\,\underline{ba}\,\underline{ac}\,\underline{cb}\,\underline{ba}\,\underline{ac}\,\underline{cb}(\underline{ba}\,\underline{ab})^\omega \\
pat(1,1,\alpha) &= (\underline{ab}\,\underline{bb}\,\underline{ba}\,\underline{ac}\,\underline{cb}) \\
pat(0,2,\alpha) &= \underline{abb}
\end{aligned}
$$

Theorem 6.5 *Let Σ be an alphabet. For all $m, n \in \mathbb{N}_0$ and every $p \in Pats(m, n, \Sigma)$ there effectively exists a formula $\varphi_p \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ such that for every $\alpha \in \Sigma^\omega$ we have that $\alpha \models \varphi_p$ iff $pat(m, n, \alpha) = p$.*

Proof: We proceed by induction on m.

- $\mathbf{m = 0}$. If $p \in Pats(0, n, \Sigma)$, then p is of the form $a_0 \dots a_n$, where each $a_i \in \Sigma$. Hence, we put

$$\varphi_p = a_0 \wedge \mathsf{X}\,(a_1 \wedge \mathsf{X}\,(a_2 \wedge \ \dots \ \wedge \mathsf{X}\,(a_{n-1} \wedge \mathsf{X}a_n) \dots)).$$

  Obviously, $\varphi_p \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^n)$. Moreover, each $\alpha \in \Sigma^\omega$ such that $\alpha \models \varphi$ starts with $a_0 \dots a_n$, which means that $pat(0, n, \alpha) = a_0 \dots a_n$. The other direction is also trivial.

- **Induction step.** Let $p \in Pats(m{+}1, n, \Sigma)$. This means that p is of the form $p = (p_1 \dots p_k)$, where each $p_i \in Pats(m, n, \Sigma)$ and $p_i \neq p_j$ for $i \neq j$. By induction hypothesis, for each $1 \leq i \leq k$ there effectively exists a formula $\varphi_{p_i} \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ which satisfies the properties of our lemma. We put

$$\varphi_p = \mathsf{G}(\varphi_{p_1} \vee \dots \vee \varphi_{p_k}) \ \wedge \ \varphi_{p_1} \ \wedge \bigwedge_{1<j\leq k} (\varphi_{p_1} \vee \dots \vee \varphi_{p_{j-1}})\,\mathsf{U}\,\varphi_{p_j}.$$

  Obviously, $\varphi_p \in \mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$. Now let $\alpha \in \Sigma^\omega$. By induction hypothesis, for all $i \in \mathbb{N}_0$ and $1 \leq j \leq k$ we have that $\alpha_i \models \varphi_{p_j}$ iff $pat(m, n, \alpha_i) = p_j$. First we prove that if $\alpha \models \varphi_p$, then $pat(m{+}1, n, \alpha) = p$. So, let $\alpha \models \varphi_p$, and let us consider the word $patword(m, n, \alpha)$. With the help of induction hypothesis, we can conclude that φ_p actually says that

  - all patterns of $Pats(m, n, \Sigma)$ which appear in $patword(m, n, \alpha)$ are among $p_1, \dots, p_n$ (this is expressed by the subformula $\mathsf{G}(\varphi_{p_1} \vee \dots \vee \varphi_{p_k})$),
  - each p_i appears in $patword(m, n, \alpha)$, and for all $1 \leq i < j \leq k$, the first occurrence of p_i precedes the first occurrence of p_j (this is expressed by the subformula $\varphi_{p_1} \ \wedge \bigwedge_{1<j\leq k}(\varphi_{p_1} \vee \dots \vee \varphi_{p_{j-1}})\,\mathsf{U}\,\varphi_{p_j}$)

  This means that $pat(m{+}1, n, \alpha) = p$ as required. The other implication (i.e. $pat(m{+}1, n, \alpha) = p$ implies $\alpha \models \varphi_p$) follows similarly. ■

Example 6.6 *Let $\alpha = abbabaaabb(ac)^\omega$. Then the formula φ_p, where $p = pat(2, 0, \alpha) = ((abc)(bac)(ac)(ca))$, is constructed as follows:*

$$\begin{aligned}
\varphi_{(abc)} &= \mathsf{G}(a \vee b \vee c) \wedge a \wedge (a\,\mathsf{U}\,b) \wedge ((a \vee b)\,\mathsf{U}\,c)\\
\varphi_{(bac)} &= \mathsf{G}(b \vee a \vee c) \wedge b \wedge (b\,\mathsf{U}\,a) \wedge ((b \vee a)\,\mathsf{U}\,c)\\
\varphi_{(ac)} &= \mathsf{G}(a \vee c) \wedge a \wedge (a\,\mathsf{U}\,c)\\
\varphi_{(ca)} &= \mathsf{G}(c \vee a) \wedge c \wedge (c\,\mathsf{U}\,a)\\
\varphi_p &= \mathsf{G}(\varphi_{(abc)} \vee \varphi_{(bac)} \vee \varphi_{(ac)} \vee \varphi_{(ca)}) \ \wedge \ \varphi_{(abc)} \ \wedge \ (\varphi_{(abc)}\,\mathsf{U}\,\varphi_{(bac)}) \wedge\\
&\quad \wedge ((\varphi_{(abc)} \vee \varphi_{(bac)})\,\mathsf{U}\,\varphi_{(ac)}) \ \wedge \ ((\varphi_{(abc)} \vee \varphi_{(bac)} \vee \varphi_{(ac)})\,\mathsf{U}\,\varphi_{(ca)})
\end{aligned}$$

Let us note that the size of φ_p for a given $p \in Pats(m, n, \Sigma)$ is exponential in the size of p. However, if φ_p is represented by a circuit (DAG), then the size of the circuit is only linear in the size of p.

Theorem 6.7 *Let Σ be an alphabet and let $m, n \in \mathbb{N}_0$. For all $\alpha, \beta \in \Sigma^\omega$ we have that α and β cannot be distinguished by any $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula if and only if $\alpha \sim_{m,n} \beta$.*

Proof: The "$\Longrightarrow$" direction follows directly from Theorem 6.5. We prove the other direction. Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. As mentioned above, we can safely assume that the X operator occurs only in subformulae of the form $\mathsf{X}\mathsf{X}\ldots\mathsf{X}a$, where a is a letter. By induction on the structure of φ we show that for every α, β such that $\alpha \sim_{m,n} \beta$ we have that $\alpha \models \varphi \iff \beta \models \varphi$.

- $\varphi = a$. It follows directly from Definition 6.3 that $\alpha(0)\ldots\alpha(n) = \beta(0)\ldots\beta(n)$. This means that $\alpha \models a \iff \beta \models a$ as required.
- **Induction step.** If $\varphi = \mathsf{X}\psi$, then $\varphi = \mathsf{X}\mathsf{X}\ldots\mathsf{X}a$, where the nesting depth of X in φ is at most n. Hence, we can argue in the same way as in the basic step. The cases when $\varphi = \neg\psi$ or $\varphi = \psi \wedge \varrho$ follow directly from induction hypothesis. Now let $\varphi = \psi \,\mathsf{U}\, \varrho$. We show that if $\alpha \models \varphi$, then also $\beta \models \varphi$. So, let $\alpha \models \varphi$. This means there is $j \in \mathbb{N}_0$ such that $\alpha_j \models \varrho$, and for every $i < j$ it holds that $\alpha_i \models \psi$. As $pat(m, n, \alpha) = pat(m, n, \beta)$, the sequence of first occurrences of letters in $patword(m-1, n, \alpha)$ is the same as in $patword(m-1, n, \beta)$. Let j' be the index of the first occurrence of a letter $pat(m-1, n, \alpha_j)$ in $patword(m-1, n, \beta)$, i.e. $pat(m-1, n, \beta_{j'}) = pat(m-1, n, \alpha_j)$. As $\varrho \in \mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^n)$, by induction hypothesis we obtain that $\beta_{j'} \models \varrho$. In the same way we can show that $\beta_{i'} \models \psi$ for every $i' < j'$, because for each such i' we have that $pat(m-1, n, \beta_{i'}) = pat(m-1, n, \alpha_i)$ for some $i < j$. This means $\beta \models \psi \,\mathsf{U}\, \rho$. ■

In other words, Theorem 6.7 says that the information about α which is relevant with respect to (in)validity of all $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae is exactly represented by $pat(m, n, \alpha)$. Thus, characteristic patterns provide a new characterization of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages.

Corollary 6.8 *An ω-language L is definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ if and only if it is closed under (m, n)-pattern equivalence.*

This characterization can be used to prove further results about LTL. In particular, a simplified form of general stuttering principle introduced in Section 5.1 follows easily from the presented results on characteristic patterns:

Lemma 6.9 *For all $m, n \in \mathbb{N}_0$, $v, w \in \Sigma^*$, $\alpha \in \Sigma^\omega$ it holds that if w is a prefix of v^ω and $|w| \geq |v| \cdot m - m + n + 1$ then $vw\alpha \sim_{m,n} w\alpha$.*

Proof: Let n, v, w, α satisfy the conditions of our lemma. We prove that $pat(m, n, vw\alpha) = pat(m, n, w\alpha)$. By induction on m.

- $\mathbf{m = 0}$. It suffices to realize that $(vw)(i) = w(i)$ for $0 \leq i \leq n$. Hence, $pat(0, n, vw\alpha) = pat(0, n, w\alpha)$.
- **Induction step (m > 0).** First we prove that for every $0 \leq i < |v|$ it holds that

$$pat(m-1, n, v_i w\alpha) = pat(m-1, n, w_i\alpha). \tag{6.1}$$

  The concatenation $v_i w$ can be seen as a concatenation $v'w_i$, where $|v'| = |v|$. Further, w_i is a prefix of $(v')^\omega$ and $|w_i| \geq |w_{|v|-1}| = |w| - |v| + 1$. Hence,

$$\begin{aligned} |w_i| &\geq |w| - |v| + 1 \\ &\geq |v| \cdot m - m + n + 1 - |v| + 1 \\ &\geq |v| \cdot (m-1) + |v| - m + n + 1 - |v| + 1 \\ &\geq |v| \cdot (m-1) - (m-1) + n + 1 \end{aligned}$$

  As $|v'| = |v|$, we obtain (6.1) by applying induction hypothesis.

  We have proven that the first $|v|$ letters of ω-words $patword(m-1, n, vw\alpha)$ and $patword(m-1, n, w\alpha)$ are the same. Further, these letters are followed by $|v|$ repeated letters in $patword(m-1, n, vw\alpha)$. As $(vw\alpha)_{2|v|} = (w\alpha)_{|v|}$, the suffixes $patword(m-1, n, vw\alpha)_{2|v|}$ and $patword(m-1, n, w\alpha)_{|v|}$ are the same. Hence, $pat(m, n, vw\alpha) = pat(m, n, w\alpha)$. ■

The conditions of Lemma 6.9 match the definition of (m, n)-redundancy of the subword v in an ω-word $vw\alpha$ given in Definition 5.6.

Theorem 6.10 *Let $m, n \in \mathbb{N}_0$, $u, v \in \Sigma^*$ and $\alpha \in \Sigma^\omega$. If v is (m, n)-redundant in $uv\alpha$, then $uv\alpha \sim_{m,n} u\alpha$.*

Proof: The theorem follows from Lemma 6.9 and the implication $\beta \sim_{m,n} \gamma \implies u\beta \sim_{m,n} u\gamma$ that can be easily proven for all $m, n \in \mathbb{N}_0$, $\beta, \gamma \in \Sigma^\omega$, and $u \in \Sigma^*$ by induction on m. ■

Theorem 6.10 is weaker than Theorem 5.9 (general stuttering theorem) which allows to add or remove infinitely many non-overlapping redundant subwords. On the other hand, Theorem 6.10 has significantly shorter proof and is still sufficient to prove all results presented in Section 5.3 including the strictness of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$, $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$, and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchies and the fact that the class of ω-languages which are definable both in $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n+1})$ is strictly larger than the class of languages definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.

6.2 Applications in model checking

In this section, the applicability of results about characteristic patterns to LTL model checking is discussed in greater detail. First of all, we introduce an algorithm deciding whether a pattern satisfies an LTL formula or not.

Definition 6.11 *Let $p \in Pats(m, n, \Sigma)$ be a pattern and $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. We say that p* satisfies *φ, written $p \models \varphi$, if for every ω-word $\alpha \in \Sigma^\omega$ we have that if $pat(m, n, \alpha) = p$, then $\alpha \models \varphi$.*

Note that Theorem 6.7 implies the following: if $p \not\models \varphi$, then for every ω-word α such that $pat(m, n, \alpha) = p$ we have $\alpha \not\models \varphi$.

Theorem 6.12 *Given an (m, n)-pattern p and an $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula φ, the problem whether $p \models \varphi$ can be decided in time $\mathcal{O}(|\varphi| \cdot |p|)$.*

```
1  proc check(φ, p, n)
2    if U-depth(φ) < mtype(p) then return(check(φ, p(0), n))
3    elsif φ = ⊤ then return(true)
4    elsif φ ∈ Σ then return(φ == p(n))
5    elsif φ = ¬φ1 then return(¬check(φ1, p, n))
6    elsif φ = φ1 ∧ φ2 then return(check(φ1, p, n) ∧ check(φ2, p, n))
7    elsif φ = Xφ1 then return(check(φ1, p, n + 1))
8    elsif φ = φ1 U φ2
9         then do
10                i := 0
11                while (i < |p|) ∧ ¬check(φ2, p(i), n) do
12                      if check(φ1, p(i), n) then i := i + 1
13                                            else i := |p|
14                      fi
15                od
16                return(i < |p|)
17           od
18   fi
```

Figure 6.1: An algorithm deciding whether $p \models \varphi$ or not.

Proof: Consider the algorithm of Figure 6.1. The procedure call $check(\varphi, p, 0)$ decides whether $p \models \varphi$ or not. The function $mtype(p)$ returns the m such that $p \in Pats(m, n, \Sigma)$. The algorithm is designed for all φ and p satisfying U-*depth*$(\varphi) \leq mtype(p)$.

The algorithm cannot assume that the X operators in φ have been pushed inside because this transformation can lead to a formula of the size

$\mathcal{O}(|\varphi|^2)$. Thus, the algorithm pushes the X operators towards letters 'virtually': the actual nesting depth of X operators is kept in the third argument of the $check$ procedure and it affects the evaluation of the subformulae of the form a (see the line 4). The correctness of the algorithm follows directly from the semantics of LTL and the idea of characteristic patterns.

The complexity of our algorithm is $\mathcal{O}(|\varphi| \cdot |p|)$ as the procedure $check$ is invoked at most once for every subformula and every subpattern. Let us note that values of U-*depth*(φ') and $mtype(p')$ for all subformulae φ' of φ and all subpatterns p' of p can be pre-calculated with the complexity $\mathcal{O}(|\varphi|+|p|)$. ■

In the rest of this section we discuss the three potential applications of characteristic patterns listed at the beginning of this chapter.

6.2.1 Decomposition technique

In this subsection we consider the variant of LTL based on atomic propositions (At) rather than letters. Hence, a formula φ is usually interpreted over infinite words over alphabet $\Sigma = 2^{At(\varphi)}$. Please note that we work with existential version of model checking problem.

Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. If our model checker fails to verify whether the system has a run satisfying φ or not (one typical reason is memory overflow), we can proceed by decomposing the formula φ in the following way:

1. First we compute the set $P = \{p \in Pats(m, n, 2^{At(\varphi)}) \mid p \models \varphi\}$.

2. Then, each $p \in P$ is translated into an equivalent LTL formula (using, for example, the algorithm of Theorem 6.5).

A simple way how to implement the first step is to compute the set $Pats(m, n, 2^{At(\varphi)})$, and then for each element p decide whether $p \models \varphi$ using the algorithm of Theorem 6.12. In practice, this could be optimized by using a more sophisticated algorithm which takes into account the structure of φ and possibly also eliminates unsatisfiable patterns. In the second step, the patterns could be alternatively translated directly into the formalism adopted in the chosen model checker (e.g. Büchi automata or alternating automata).

Example 6.13 *We illustrate the decomposition technique on a formula* $\varphi = \mathsf{FG}\neg q$ *which is the negation of a typical liveness property* $\mathsf{GF}q$*. The alphabet is* $\Sigma = 2^{\{q\}} = \{\{q\}, \emptyset\}$*. To simplify our notation, we use* A *and* B *to abbreviate* $\{q\}$ *and* $\emptyset$*, respectively. The elements of* $Pats(2, 0, \{A, B\})$ *are listed below (unsatisfiable*

patterns have been eliminated). All patterns which satisfy φ are listed in the right column.

$((A))$	$((B))$
$((BA)(A))$	$((AB)(B))$
$((AB)(BA))$	$((BA)(AB)(B))$
$((BA)(AB))$	$((AB)(BA)(B))$
$((AB)(BA)(A))$	
$((BA)(AB)(A))$	

The formulae corresponding to the patterns of the right column are listed below.[3]

$((B))$	$\psi_1 = \mathsf{G}\neg q$
$((AB)(B))$	$\psi_2 = q \wedge q \,\mathsf{U}\, \mathsf{G}\neg q$
$((BA)(AB)(B))$	$\psi_3 = \neg q \wedge \mathsf{F}(q \wedge \mathsf{F}\neg q) \wedge \mathsf{FG}\neg q$
$((AB)(BA)(B))$	$\psi_4 = q \wedge \mathsf{F}(\neg q \wedge \mathsf{F}q) \wedge \mathsf{FG}\neg q$

So, the formula φ is decomposed into an equivalent disjunction $\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$.

Thus, the original question whether the system has a run satisfying φ is decomposed into k questions of the same type. These can be solved using standard model checkers.

We illustrate potential benefits of this method in the context of automata-based approach to model checking (see Section 2.4). Here the formula φ is translated into a Büchi automaton A_φ accepting the ω-language $L(\varphi)$. Then, the model checking algorithm computes another Büchi automaton called *product automaton*, which accepts exactly those runs of the verified system which are accepted by A_φ as well. The model checking problem is thus reduced to the problem whether the language accepted by the product automaton is empty or not. The bottleneck of this approach is the size of the product automaton.

Example 6.14 *Let us suppose that a given model checking algorithm does not manage to check the formula φ of Example 6.13. The subtasks given by the ψ_i formulae constructed in Example 6.13 can be more tractable. Some of the reasons are illustrated below.*

- *The size of the Büchi automaton for ψ_i can be smaller than the size of A_φ. In Example 6.13, this is illustrated by formula ψ_1 (see Figure 6.2). The corresponding product automaton is then smaller as well.*
- *The size of the product automaton constructed for ψ_i can be smaller than the one for φ even if the size of A_{ψ_i} is larger than the size of A_φ. In Example 6.13, this is illustrated by the formula ψ_2; the automata for φ and ψ_2 are almost the same (see Figure 6.2), but the product automaton for ψ_2 can be much smaller as indicated in Figure 6.3.*

[3] For notation convenience, we simplified the formulae obtained by running the algorithm of Theorem 6.5 into a more readable (but equivalent) form.

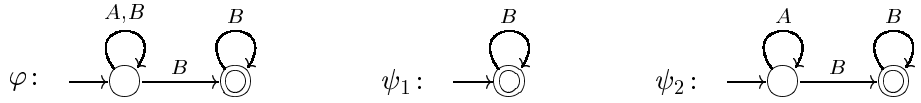

Figure 6.2: Büchi automata corresponding to formulae φ, ψ_1, and ψ_2 of Example 6.13.

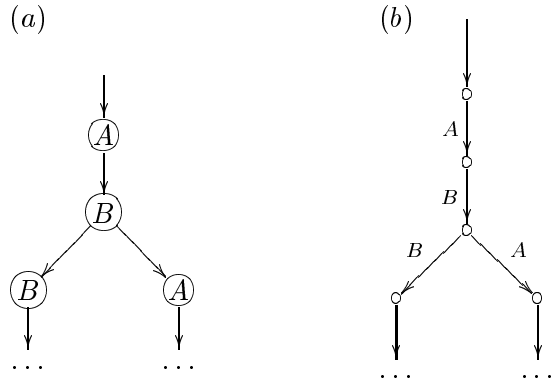

Figure 6.3: An example of a system to be verified (a) and product automata (b) and (c) corresponding to φ and ψ_2 of Example 6.13, respectively.

It is of course possible that some of the ψ_i formulae in the constructed decomposition remain intractable. Let ψ_i be such an intractable formula. Then ψ_i can be further decomposed by a technique called *refinement* (since ψ_i corresponds to a unique pattern $p_i \in Pats(m, n, 2^{At(\varphi)})$, we can equivalently consider pattern refinement). There are two basic ways how to refine the pattern p_i. The idea of the first method is to compute the set of (m', n')-patterns, where $m' \geq m$ and $n' \geq n$, and identify all patterns that satisfy the formula ψ_i.

Example 6.15 *The formula ψ_3 of Example 6.13 corresponding to the $(2, 0)$-pattern $((BA)(AB)(B))$ can be refined into two $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X}^0)$ formulae given by the $(3, 0)$-patterns*

$$(((BA)(AB)(B))((AB)(B))((B))),$$
$$(((BA)(AB)(B))((AB)(BA)(B))((AB)(B))((B))).$$

The other refinement method is based on enlarging the alphabet before computing the patterns. We simply expand the set $At(\varphi)$ with a new atomic proposition. The choice of the new atomic proposition is of course important. By a "suitable" choice we mean a choice which leads to a convenient split of system's runs into more manageable units. An interesting problem

(which is a potential topic for future work) is whether suitable new propositions can be identified effectively.

Example 6.16 *Let us consider the formula ψ_2 of Example 6.13 corresponding to the $(2,0)$-pattern $((AB)(B))$. The original set of atomic propositions $At(\varphi) = \{q\}$ generates the alphabet $\Sigma = \{A, B\}$, where $A = \{q\}, B = \emptyset$. If we enrich the set of atomic propositions with r, we get a new alphabet $\Sigma' = \{C, D, E, F\}$, where $C = \{q, r\}, D = \{q\}, E = \{r\}, F = \emptyset$. Hence, the original letters A, B correspond to the pairs of letters C, D and E, F, respectively. Thus, the formula ψ_2 is refined into $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ formulae given by the $(2,0)$-patterns*

$$
\begin{array}{l}
((CE)(E)) \\
((CDE)(DE)(E)) \\
((CDE)(DCE)(CE)(E)) \\
((CDE)(DCE)(DE)(E)) \\
((CEF)(EF)(FE)) \\
((CEF)(EF)(FE)(E)) \\
((CEF)(EF)(FE)(F)) \\
((CDEF)(DEF)(EF)(FE)) \\
((CDEF)(DEF)(EF)(FE)(E)) \\
((CDEF)(DEF)(EF)(FE)(F)) \\
((CDEF)(DCEF)(CEF)(EF)(FE)) \\
((CDEF)(DCEF)(CEF)(EF)(FE)(E)) \\
((CDEF)(DCEF)(CEF)(EF)(FE)(F)) \\
((CDEF)(DCEF)(DEF)(EF)(FE)) \\
((CDEF)(DCEF)(DEF)(EF)(FE)(E)) \\
((CDEF)(DCEF)(DEF)(EF)(FE)(F))
\end{array}
$$

and all those patterns which can be obtained from the above given ones by either exchanging the letters C, D, or exchanging the letters E, F, or by both exchanges. Hence, the formula ψ_2 is refined into a disjunction of $16 \cdot 4 = 64$ formulae.

Some of the subtasks obtained by refining intractable subtasks can be tractable. Others can be refined again and again. Observe that even if we solve only some of the subtasks, we still obtain a new piece of relevant knowledge about the system – we know that if the system has a run satisfying φ, then the run satisfies one of the formulae corresponding to the subtasks we did not manage to solve. Hence, we can (at least) classify and repeatedly refine the set of "suspicious" runs.

We finish this subsection by listing the benefits and drawbacks of the presented method.

+ The subtasks are formulated as standard model checking problems. Therefore, the method can be combined with all existing algorithms and heuristics.

+ With the help of the method, we can potentially verify some systems which are beyond the reach of existing model checkers.

+ Even if it is not possible complete the verification task, we get partial information about the structure of potential (undiscovered) runs satisfying φ. We also know which runs of the system have been successfully verified.

+ The subtasks can be solved simultaneously in a distributed environment with a very low communication overhead.

+ When we verify more formulae on the same system, the subtasks occurring in decompositions of both formulae are solved just once.

– Calculating the decomposition of a given formula can be expensive. On the other hand, this is not critical for formulae with small number of atomic propositions and small nesting depths of U and X.

– Runtime costs of the proposed algorithm are high. It can happen that all subtasks remain intractable even after several refinement rounds and we get no new information at all.

6.2.2 Model checking a path using patterns

Model checking a path (see Subsection 2.4.2) has been identified as another application of characteristic patterns. The problem is to decide whether a given loop uv^ω satisfies a given formula φ or not. It the following we deal with the case when φ is a formula of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$. This version of the problem can be solved in time $\mathcal{O}(|uv| \cdot |\varphi|)$. More information about complexity of model checking a path Section 4.2. Here we present a new algorithm based on characteristic patterns and argue that our algorithm can be more efficient in some cases.

Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula and uv^ω be a loop, where $u, v \in \Sigma^*$ and $v \neq \varepsilon$. The algorithm first computes a pattern $pat(m, n, uv^\omega)$ and then it decides whether the pattern satisfies φ. First we focus on the pattern extraction.

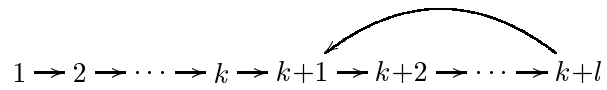

Figure 6.4: A finite-state system with one infinite path.

Let $k = |u|$ and $l = |v|$. The loop can be represented by the finite structure given in Figure 6.4 and a function L labelling each state of the structure

with the corresponding letter of the loop, i.e.

$$L(i) = \begin{cases} u(i-1) & \text{if } 1 \leq i \leq k, \\ v(i-k-1) & \text{if } k+1 \leq i \leq k+l. \end{cases}$$

By $s(i)$ we denote a *successor* of a state i defined by arrow leading from i.

```
1 for i := 1 to k + l do
2     L'(i) := L(i)L(s(i))L(s^2(i)) ... L(s^n(i))
3 od
4 for i := 1 to m do
5     L'(k+l) := the parenthesized word obtained from
                 L'(k+l)L'(k+1)L'(k+2) ... L'(k+l-1)
                 by deletion of all repeated letters
6     for j := k + l - 1 downto 1 do
7         L'(j) := the word L'(j+1) with the letter L'(j)
                   added to the beginning and without
                   any repetition of this letter
8     od
9 od
```

Figure 6.5: An algorithm for (m, n)-pattern extraction.

The pattern extraction algorithm given in Figure 6.5 computes a new labelling function L'. The desired pattern $pat(m, n, uv^\omega)$ is stored in $L'(1)$ after the algorithm terminates. The algorithm is based directly on the definition of characteristic patterns. After the i-th iteration of the second for-loop, the labels stored in $L'(1) \ldots L'(k+l)$ describe the ω-word $patword(i, n, uv^\omega)$. More precisely, $patword(i, n, uv^\omega) = L'(1) \ldots L'(k) \, (L'(k+1) \ldots L'(k+l))^\omega$. The time complexity of this algorithm is $\mathcal{O}(|uv| \cdot (n + m \cdot S(m, n, \Sigma)))$, where $S(m, n, \Sigma)$ is the maximal size of a pattern in $Pats(m, n, \Sigma)$ (as given in Lemma 6.2).

Due to Theorem 6.12, the problem whether $pat(m, n, uv^\omega) \models \varphi$ can be solved in $\mathcal{O}(S(m, n, \Sigma) \cdot |\varphi|)$ time. Hence, the algorithm needs $\mathcal{O}(|uv|(n + m \cdot S(m, n, \Sigma)) + S(m, n, \Sigma) \cdot |\varphi|)$ time in total. In the light of this estimation, our algorithm seems to be only worse than the bilinear CTL-like algorithm. However, if we bound the parameters m, n, and $|\Sigma|$ by constants (this is justifiable as these are usually "small") then our algorithm needs only $\mathcal{O}(|uv| + |\varphi|)$ time, while the CTL-like algorithm still requires $\mathcal{O}(|uv| \cdot |\varphi|)$ time. In other words, our algorithm is better in situations when m, n and $|\Sigma|$ are small, and $|uv|$ is large. Then it pays to extract the characteristic pattern from the loop and check the formulae directly on the pattern rather than on the loop itself.

6.2.3 Partial order reduction using patterns

Characteristic patterns can also be applied in a more conventional way: to reduce Kripke structures. Intuitively, if we want to decide whether an $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula is valid for a Kripke stricture, we do not need to examine all runs of the structure. In fact, it is sufficient to consider only the subset of all runs such that for every run of the structure there is a run in the subset with the same (m, n)-pattern.

In Section 5.4 we have shown that general stuttering theorem allows to produce smaller reduced Kripke structures than those produced by standard partial order reduction methods. While general stuttering principle formulates just a sufficient condition for two words to be indistinguishable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, Theorem 6.7 says that two words are indistinguishable by any $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula if and only if the words have the same (m, n)-pattern. Hence, reduction based on characteristic patterns can produce smaller Kripke structures than the reduction based on general stuttering (and thus also smaller than the standard partial order reduction based on stuttering).

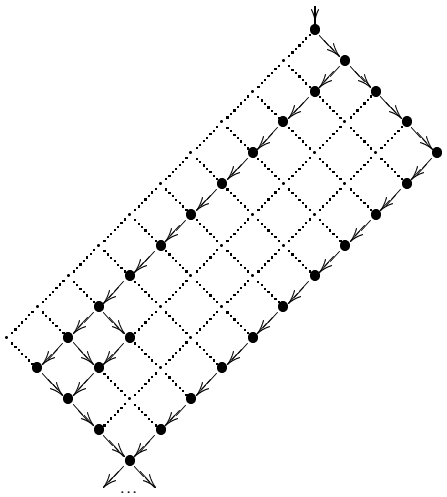

Figure 6.6: The Kripke structure reduced with use of characteristic patterns.

Example 6.17 *Similarly to Example 5.22, we consider the problem to check whether the Kripke structure defined in Example 2.25 and depicted in Figure 2.3 satisfies a specification formula $\varphi \in \mathrm{LTL}(\mathsf{U}^1)$ with atomic propositions dependent just on the value of variable* x*. Thanks to Theorem 6.7, it is sufficient to check the reduced structure given by Figure 6.6. The reduced structure is notably smaller than the structures depicted in Figures 5.6 and 2.4.*

As in the case of reduction based on letter stuttering or general stuttering, we have to emphasize that the reduced Kripke structure has been made by hand and no practical algorithm for partial order reduction using patterns has been proposed so far. It is another topic for our future work.

6.3 Additional notes

The results presented in this section do not hold for finite words. The argument is that an $\mathrm{LTL}(\mathsf{U}, \mathsf{X}^n)$ formula can express a condition on last n letters of a finite word. For example, a finite word satisfies a formula $\mathsf{F}(a \wedge \neg \mathsf{X}\top)$ if and only if the last letter of the word is a. A modification of characteristic patterns for finite words is a topic for future work.

Another interesting question (which is left open) is whether one could use characteristic patterns to prove the decidability of fragments of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X})$ over ω-words.

Chapter 7

Deeper connections to alternating automata

The chapter is based on the results presented in [PS04].

An automata-theoretic approach to the study of temporal logics proved to be very fruitful. The best example is the well-known fact that each LTL formula can be translated into nondeterministic Büchi automaton that accepts exactly the infinite words satisfying the formula [WVS83, VW94]. The translation was published in 1983 and soon became one of the cornerstones of the automata-based model checking of LTL properties [VW86]. Approximately at the same time it was also shown that there are Büchi automata accepting languages that are not definable by any LTL formula [Wol83].

Later on, alternating 1-weak Büchi automata (or A1W automata for short, also known as alternating linear automata or very weak alternating automata) have been identified as the type of automata with the same expressive power as LTL. In Section 7.1 we recall two results showing the equivalence of LTL and A1W. The first one is a translation of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ formulae into equivalent A1W automata introduced by Muller, Saoudi, and Schupp [MSS88]. This translation produces A1W automata with number of states linear in the length of input formula contrary to the mentioned translation into Büchi automata where the number of states is exponential. The translation of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ into A1W automata has been recently used to built new and more efficient algorithms for translation of $\mathrm{LTL}(\mathsf{U},\mathsf{X})$ into nondeterministic Büchi automata [GO01, Tau03][1]. The second result is a translation of A1W automata into equivalent LTL formulae presented independently by Rohde [Roh97] and Löding and Thomas [LT00].

In the light of research on LTL fragments, it is natural to ask for classes of A1W automata with the same expressive power as the studied LTL frag-

[1] In fact, the paper [GO01] employs alternating 1-weak co-Büchi automata. However, Büchi and co-Büchi acceptance conditions are expressively equivalent for alternating 1-weak automata.

ments. It turns out that the translation of A1W automata into LTL mentioned above is not appropriate for study of such automata classes as it wastes temporal operators. For example, the automaton corresponding to the formula $a \mathsf{U} (b \wedge (b \mathsf{U} c))$ is translated into formula $a \mathsf{U} (b \wedge \mathsf{X}(b \mathsf{U} c))$. In Section 7.2 we present an improved translation of A1W automata into equivalent LTL formulae. Our translation reduces the nesting depth of X operators and prefers the use of less expressive temporal operators F or G instead of U operator. We prove that for an A1W automaton produced by the standard translation of a given $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula our translation provides a formula from the same fragment.

In Section 7.3 we identify classes of A1W automata defining the same language classes as some recognized fragments of LTL, namely the fragments of the until-release hierarchy [ČP03] and fragments of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n\mathsf{F}^k)$ where $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$.

Section 7.4 indicates several topics for future work.

For definition of alternating 1-weak Büchi automata we refer to Subsection 2.3.3. In that subsection there are also definitions of some terms and notation used in this chapter, namely the definition of a state with a *loop* and notation $A(p)$, $Succ(p)$, $Succ'(p)$, and $p \xrightarrow{a} S$.

7.1 Equivalence of LTL and A1W automata

In this section we recall the standard translation of $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formulae to A1W automata [MSS88] (marked as LTL→A1W) and a translation of A1W automata to $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ presented recently in [LT00] (and marked as A1W→LTL here). The latter translation has been independently introduced by Rohde in [Roh97].

7.1.1 LTL→A1W translation

Let φ be an $\mathrm{LTL}(\mathsf{U}, \mathsf{X})$ formula and Σ be an alphabet. The formula can be translated into an automaton A such that $L(A) = L^{\Sigma}(\varphi)$. The automaton A is defined as $A = (\Sigma, Q, q_\varphi, \delta, F)$, where

- the states $Q = \{q_\psi, q_{\neg\psi} \mid \psi \text{ is a subformula of } \varphi\}$ correspond to the subformulae of φ and their negations,
- the transition function δ is defined inductively in the following way:

$$
\begin{array}{l}
\delta(q_\top, a) = \top \\
\delta(q_a, b) = \top \text{ if } a = b,\ \delta(q_a, b) = \bot \text{ otherwise} \\
\delta(q_{\neg\psi}, a) = \overline{\delta(q_\psi, a)} \\
\delta(q_{\psi\wedge\rho}, a) = \delta(q_\psi, a) \wedge \delta(q_\rho, a) \\
\delta(q_{\mathsf{X}\psi}, a) = q_\psi \\
\delta(q_{\psi\mathsf{U}\rho}, a) = \delta(q_\rho, a) \vee (\delta(q_\psi, a) \wedge q_{\psi\mathsf{U}\rho})
\end{array}
$$

where $\overline{\tau}$ denotes the positive boolean formula dual to τ defined by induction on the structure of τ as follows:

$$\begin{array}{lll} \overline{\top} = \bot & \overline{q_{\neg\psi}} = q_{\psi} & \overline{\sigma \wedge \xi} = \overline{\sigma} \wedge \overline{\xi} \\ \overline{\bot} = \top & \overline{q_{\psi}} = q_{\neg\psi} & \overline{\sigma \vee \xi} = \overline{\sigma} \vee \overline{\xi} \end{array}$$

- the set of accepting states $F = \{q_{\neg(\psi \mathsf{U} \rho)} \mid \psi \mathsf{U} \rho \text{ is a subformula of } \varphi\}$.

We use the notation $A^{\Sigma}(\varphi)$ for the automaton given by the translation of an LTL formula φ with respect to an alphabet Σ. The number of states of the automaton $A^{\Sigma}(\varphi)$ is clearly linear in the length of φ.

For example, the translation applied on the formula $\varphi = (a \mathsf{U} b) \wedge \mathsf{FG}c$ and the alphabet $\Sigma = \{a, b, c\}$ produces the automaton depicted on Figure 2.2, where p, q_1, q_2, q_3 stand for $q_{\varphi}, q_{a\mathsf{U}b}, q_{\mathsf{FG}c}, q_{\mathsf{G}c}$, respectively.

7.1.2 A1W→LTL translation

Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For each $p \in Q$ we define an LTL formula φ_p such that $L^{\Sigma}(\varphi_p) = L(A(p))$. (in particular $L^{\Sigma}(\varphi_{q_0}) = L(A)$). The definition proceeds by induction respecting the ordering of states; the formula φ_p employs formulae of the form φ_q where $q \in Succ'(p)$. This is the point where the 1-weakness of the automaton is used. To illustrate the inductive step of the translation, let us consider the situation depicted on Figure 7.1. The formula corresponding to state p is $\varphi_p = (a \wedge \mathsf{X}\varphi_q) \mathsf{U} (b \wedge \mathsf{X}\varphi_r)$.

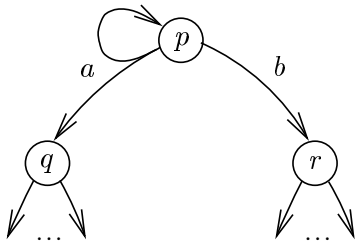

Figure 7.1: Part of an automaton translated into the formula $\varphi_p = (a \wedge \mathsf{X}\varphi_q) \mathsf{U} (b \wedge \mathsf{X}\varphi_r)$.

Before we give a formal definition of φ_p, we introduce some auxiliary formulae. Let $a \in \Sigma$ be a letter and $S \subseteq Q$ be a set of states. The formula

$$\theta(a, S) = a \wedge \bigwedge_{q \in S} \mathsf{X}\varphi_q$$

intuitively corresponds to a situation when automaton makes transition under a into the set of states S. Formulae σ_p and ξ_p defined as

$$\sigma_p = \bigvee_{\substack{p \xrightarrow{a} S \\ p \in S}} \theta(a, S \setminus \{p\}) \qquad \xi_p = \bigvee_{\substack{p \xrightarrow{a} S \\ p \notin S}} \theta(a, S)$$

intuitively correspond to all transitions leading from state p; σ_p covers the transitions with a loop (i.e. the transitions leading to the set of states containing p) while ξ_p cover the others. The definition of φ_p then depends on whether p is an accepting state or not.

$$\varphi_p = \begin{cases} \sigma_p \,\mathsf{U}\, \xi_p & \text{if } p \notin F \\ (\sigma_p \,\mathsf{U}\, \xi_p) \vee \mathsf{G}\sigma_p & \text{if } p \in F \end{cases}$$

The proof of the correctness of this translation can be found in [LT00]. Given an A1W automaton A with an initial state q_0, by $\varphi(A)$ we denote the formula $\varphi(A) = \varphi_{q_0}$.

7.2 Improving A1W→LTL translation

The A1W→LTL translation presented in the previous section is not optimal: it wastes temporal operators. The main problem of the translation is that for each successor $q \in Succ'(p)$ of a state p the formula φ_p contains a subformula $\mathsf{X}\varphi_q$ even if the X operator is not needed. This can be illustrated by an automaton A on Figure 7.2. The automaton is created by translation of the

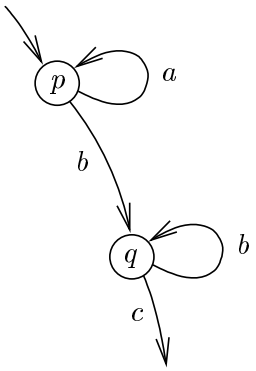

Figure 7.2: The automaton for the formula $a \,\mathsf{U}\, (b \wedge b \,\mathsf{U}\, c)$.

formula $a \,\mathsf{U}\, (b \wedge b \,\mathsf{U}\, c)$. The A1W→LTL translation provides an equivalent formula $\varphi(A) = a \,\mathsf{U}\, (b \wedge \mathsf{X}(b \,\mathsf{U}\, c))$ in spite of it.

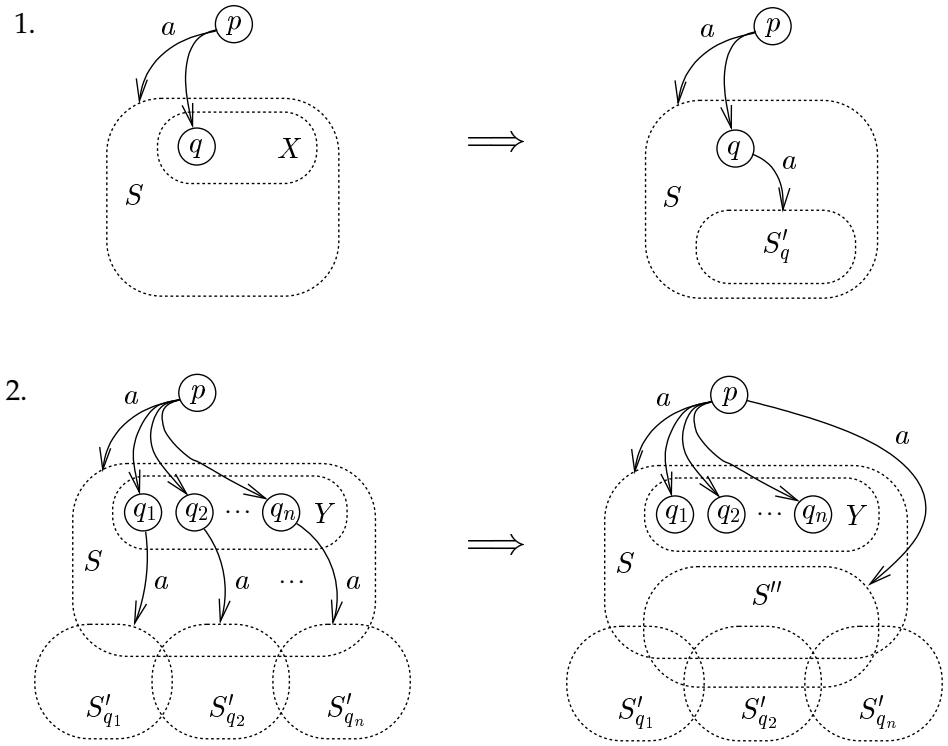

Figure 7.3: The conditions for X-freeness.

Let $p \xrightarrow{a} S$ be a transition and $X \subseteq S$. We now formulate conditions that are sufficient to omit the X operator in front of φ_q (for every $q \in X$) in a subformula of φ_p corresponding to the transition $p \xrightarrow{a} S$. A set X satisfying these conditions is called X*-free*.

Definition 7.1 *Let $p \xrightarrow{a} S$ be a transition of an automaton A. A set $X \subseteq S \setminus \{p\}$ is said to be* X-free for $p \xrightarrow{a} S$ *if following conditions hold.*

1. *For each $q \in X$ there is $S'_q \subseteq S$ such that $q \xrightarrow{a} S'_q$.*

2. *Let $Y \subseteq X$ and for each $q \in Y$ let $S'_q \subseteq Q$ be a set satisfying $q \xrightarrow{a} S'_q$ and $q \notin S'_q$. Then there exists a set $S'' \subseteq (S \setminus Y) \cup \bigcup_{q \in Y} S'_q$ satisfying $p \xrightarrow{a} S''$.*

The conditions for X-freeness are illustrated by Figure 7.3. Please note that it can be the case that $p \in S$. Further, in the first condition it can be the case that $q \in S'_q$.

It is easy to see that empty set is X-free for every transition. Further, every subset of a X-free set for a transition is a X-free set for the transition as well. On the other hand, Figure 7.4 demonstrates that the union of two

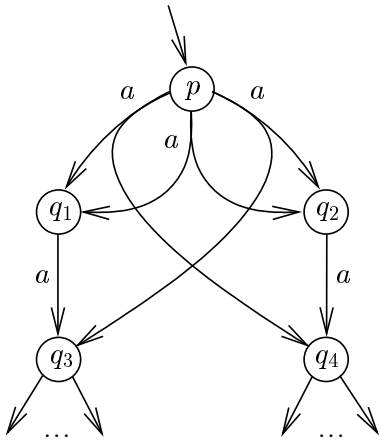

Figure 7.4: The sets $\{q_1\}, \{q_2\}$ are X-free for $p \xrightarrow{a} \{q_1, q_2\}$ while the set $\{q_1, q_2\}$ is not.

X-free sets need not be X-free; in the automaton indicated on the figure, the sets $\{q_1\}, \{q_2\}$ are X-free for $p \xrightarrow{a} \{q_1, q_2\}$ while the set $\{q_1, q_2\}$ is not.

Let Xfree be an arbitrary but fixed function assigning to each transition $p \xrightarrow{a} S$ a set that is X-free for $p \xrightarrow{a} S$. We now introduce an improved A1W→LTL translation. Roughly speaking, the translation omits the X operators in front of subformulae which correspond to the states in X-free sets given by the function Xfree. Thereafter we prove that this translation remains correct.

The improved A1W→LTL translation exhibits similar structure as the original one. Instead of formulae of the form $\theta(a, S)$ representing a transition under a leading from an arbitrary state p to S, we define a specialized formula $\theta'_p(a, S)$ for each transition $p \xrightarrow{a} S$.

$$\theta'_p(a,S) = a \;\wedge \bigwedge_{\substack{q \,\in\, S \,\smallsetminus\, \mathsf{Xfree}(p \xrightarrow{a} S) \\ q \neq p}} \mathsf{X}\varphi'_q \;\wedge \bigwedge_{q \,\in\, \mathsf{Xfree}(p \xrightarrow{a} S)} \varphi'_q$$

$$\sigma'_p = \bigvee_{\substack{p \xrightarrow{a} S \\ p \,\in\, S}} \theta'_p(a,S) \qquad\qquad \xi'_p = \bigvee_{\substack{p \xrightarrow{a} S \\ p \,\notin\, S}} \theta'_p(a,S)$$

We also identify some cases when U can be replaced by "weaker" operators F or G in a formula φ'_p. To this end we define two special types of states. A state p is of the F-*type* if there is a transition $p \xrightarrow{a} \{p\}$ for every $a \in \Sigma$. A state p is of the G-*type* if every transition of the form $p \xrightarrow{a} S$ satisfies $p \in S$.

$$\varphi'_p = \begin{cases} \xi'_p & \text{if } p \notin \mathit{Succ}(p) \\ \bot & \text{if } p \in \mathit{Succ}(p),\ p \notin F,\ p \text{ is of G-type} \\ \mathsf{F}\xi'_p & \text{if } p \in \mathit{Succ}(p),\ p \notin F,\ p \text{ is of F-type and not of G-type} \\ \sigma'_p \,\mathsf{U}\, \xi'_p & \text{if } p \in \mathit{Succ}(p),\ p \notin F,\ p \text{ is not of F-type or G-type} \\ \top & \text{if } p \in \mathit{Succ}(p),\ p \in F,\ p \text{ is of F-type} \\ \mathsf{G}\sigma'_p & \text{if } p \in \mathit{Succ}(p),\ p \in F,\ p \text{ is of G-type and not of F-type} \\ (\sigma'_p \,\mathsf{U}\, \xi'_p) \vee \mathsf{G}\sigma'_p & \text{if } p \in \mathit{Succ}(p),\ p \in F,\ p \text{ is not of F-type or G-type} \end{cases}$$

The new cases in the definition of φ'_p make only a cosmetic change comparing to the original A1W→LTL translation. First, we add a case for states without any loop. This change does not influence the correctness of the translation as the condition $p \notin \mathit{Succ}(p)$ says that $\sigma'_p = \bot$ and therefore $\varphi'_p = \xi_p$ is equivalent to $\sigma'_p \,\mathsf{U}\, \xi'_p$ as well as to $(\sigma'_p \,\mathsf{U}\, \xi'_p) \vee \mathsf{G}\sigma'_p$. Further, it is easy to check that if a state p is of F-type then $\sigma'_p \iff \top$ and if p is of G-type then $\xi'_p = \bot$. Hence, all cases for $p \in \mathit{Succ}(p)$ and $p \notin F$ are equivalent to $\sigma'_p \,\mathsf{U}\, \xi'_p$ and all cases for $p \in \mathit{Succ}(p)$ and $p \in F$ are equivalent to $(\sigma'_p \,\mathsf{U}\, \xi'_p) \vee \mathsf{G}\sigma'_p$.

Before we show that the improved translation is equivalent to the original one (and thus also correct), we prove two auxiliary lemmata.

Lemma 7.2 *For each transition $p \xrightarrow{a} S$ of an A1W automaton A the implication $\theta(a, S \smallsetminus \{p\}) \Longrightarrow \theta'_p(a, S)$ holds assuming that $\varphi_q \iff \varphi'_q$ for each $q \in \mathit{Succ}'(p)$.*

Proof: Due to the definitions of $\theta(a, S \smallsetminus \{p\})$ and $\theta'_p(a, S)$ and the assumption of the lemma, it is sufficient to show for each ω-word $\alpha \in \Sigma^\omega$ that

$$\text{if } q \in \mathsf{Xfree}(p \xrightarrow{a} S) \text{ and } \alpha \models \theta(a, S \smallsetminus \{p\}) \text{ then } \alpha \models \varphi_q.$$

The first condition for X-freeness gives us that there is $S'_q \subseteq S$ such that $q \xrightarrow{a} S'_q$. From the 1-weakness of the automaton we have that $p \notin S'_q$. Thus, $S'_q \subseteq S \smallsetminus \{p\}$ and $\alpha \models \theta(a, S \smallsetminus \{p\})$ implies $\alpha \models \theta(a, S'_q \smallsetminus \{q\})$. As $\alpha \models \theta(a, S'_q \smallsetminus \{q\})$ and $q \xrightarrow{a} S'_q$ we have that either $q \in S'_q$ and then $\alpha \models \sigma_q$, or $q \notin S'_q$ and then $\alpha \models \xi_q$. Anyway, $\alpha \models \sigma_q \vee \xi_q$ holds. At the same time $\alpha \models \theta(a, S \smallsetminus \{p\})$ implies $\alpha \models \mathsf{X}\varphi_q$. We are done as $\alpha \models \sigma_q \vee \xi_q$ and $\alpha \models \mathsf{X}\varphi_q$ imply $\alpha \models \varphi_q$. ■

Lemma 7.3 *Let $p \xrightarrow{a} S$ be a transition of an A1W automaton A. Then $\theta'_p(a, S) \Longrightarrow \xi_p \vee \sigma_p$ assuming that $\varphi_q \iff \varphi'_q$ for each $q \in \mathit{Succ}'(p)$. Moreover, if $p \notin S$ then $\theta'_p(a, S) \Longrightarrow \xi_p$ on the same assumption.*

Proof: Let us suppose that $\alpha \in \Sigma^\omega$ is an ω-word such that $\alpha \models \theta'_p(a, S)$. Due to the assumption of the lemma the formula $\theta'_p(a, S)$ is equivalent to

$$a \wedge \bigwedge_{\substack{q \in S \setminus \mathsf{Xfree}(p \xrightarrow{a} S) \\ q \neq p}} \mathsf{X}\varphi_q \wedge \bigwedge_{q \in \mathsf{Xfree}(p \xrightarrow{a} S)} \varphi_q.$$

Obviously $\varphi_q \Longrightarrow \xi_q \vee \mathsf{X}\varphi_q$. Let Y be the set

$$Y = \{q \in \mathsf{Xfree}(p \xrightarrow{a} S) \mid \alpha \models \xi_q\}.$$

For each $q \in Y$, the definition of the formula ξ_q and the assumption $\alpha \models a$ (due to $\alpha \models \theta'_p(a, S)$) imply that there exists a set S'_q such that $q \xrightarrow{a} S'_q$, $q \notin S'_q$, and $\alpha \models \theta(a, S'_q)$. The second condition for X-freeness gives us that there exists a set $S'' \subseteq (S \setminus Y) \cup \bigcup_{q \in Y} S'_q$ satisfying $p \xrightarrow{a} S''$. As $\alpha \models \mathsf{X}\varphi_q$ for every $q \in S \setminus Y$ and $\alpha \models \theta(a, S'_q)$ for each $q \in Y$, we get that $\alpha \models \theta(a, S'' \setminus \{p\})$ as well. To sum up, we have a set S'' such that $p \xrightarrow{a} S''$ and $\alpha \models \theta(a, S'' \setminus \{p\})$. If $p \in S''$ then $\alpha \models \sigma_p$. Moreover, 1-weakness of the automaton implies that $p \notin S'_q$ and therefore $p \in S$. Finally, if $p \notin S''$ then $\alpha \models \xi_p$. ■

We are now ready to prove the correctness of the improved translation.

Theorem 7.4 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For every state $p \in Q$ the equation $L(A(p)) = L^\Sigma(\varphi'_p)$ holds.*

Proof: We show that $\varphi_p \Longleftrightarrow \varphi'_p$ holds for every $p \in Q$. The proof proceeds by induction with respect to the ordering on Q. If $Succ'(p) = \emptyset$ then empty set is the only X-free set for any transition leading from p. Hence φ_p and φ'_p are equivalent.

Let us now assume that the equivalence holds for every $q \in Succ'(p)$. The Lemma 7.2 implies $\varphi_p \Longrightarrow \varphi'_p$. Lemma 7.3 gives us that ξ'_p implies ξ_p, and σ'_p implies $\xi_p \vee \sigma_p$. As an immediate consequence we get $\varphi'_p \Longrightarrow \varphi_p$. ■

Let A be an A1W automaton and q_0 its initial state. By $\varphi^{\mathsf{Xfree}}(A)$ we denote the formula φ'_{q_0} given by the improved translation which is determined by the function Xfree.

After it has been proven that the improved translation remains correct, it is only natural to examine the "quality" of formulae it produces. The improved translation has been motivated by the observation that the standard one wastes X operators. Therefore, we show that the improved translation allows to translate an automaton $A^\Sigma(\varphi)$ derived from a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ back into a formula from $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. In order to do so,

we define two metrics for A1W automata, namely *loop-height* and X*-height*, and show that an automaton A with loop-height m and X-height n can be translated into a formula form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. Then we prove that an automaton $A^\Sigma(\varphi)$ given by the standard LTL→A1W translation of a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ has loop-height and X-height at most m and n, respectively. These relations also enable us to define some of the studied LTL fragments via A1W automata.

Definition 7.5 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For each state $p \in Q$ we inductively define its* loop-height *and* X-height *(denoted by $lh(p)$ and $\mathsf{X}h(p)$ respectively) as*

$$lh(p) = \begin{cases} \max\{lh(q) \mid q \in Succ'(p)\} + 1 & \text{if } p \in Succ(p), \\ \max\{lh(q) \mid q \in Succ'(p)\} & \text{otherwise}, \end{cases}$$

$$\mathsf{X}h(p) = \max_{p \xrightarrow{a} S} \{ \min_{X \text{ is } \mathsf{X}\text{-free for } p \xrightarrow{a} S} \{need\mathsf{X}(p \xrightarrow{a} S, X)\} \},$$

where maximum over empty set is 0 *and*

$$need\mathsf{X}(p \xrightarrow{a} S, X) = \max(\{\mathsf{X}h(q) \mid q \in X\} \cup \{\mathsf{X}h(q)+1 \mid q \in S \setminus X, q \neq p\}).$$

We also define loop-height *and* X-height *of the automaton A as the* loop-height *and* X-height *of its initial state, i.e. $lh(A) = lh(q_0)$ and $\mathsf{X}h(A) = \mathsf{X}h(q_0)$.*

Intuitively, loop-height of an automaton A holds the maximal height of states of A with a loop. The X-height counts the minimal nesting depth of X operators achievable by the improved translation; the definition consider the minimum over all choices of X-free sets.

Theorem 7.6 *Let A be an A1W automaton. There exists a function* Xfree *such that $\varphi^{\mathsf{Xfree}}(A) \in \mathrm{LTL}(\mathsf{U}^{lh(A)}, \mathsf{X}^{\mathsf{X}h(A)})$.*

Proof: Please note that U*-depth*$(\varphi^{\mathsf{Xfree}}(A))$ does not depend on the choice of the function Xfree. Contrary to the U*-depth*, the X*-depth* of the resulting formula depends on the function Xfree. The function Xfree satisfying X*-depth*$(\varphi^{\mathsf{Xfree}}(A)) = \mathsf{X}h(A)$ can be derived directly from the definition of X-height; for every transition $p \xrightarrow{a} S$ we set $\mathsf{Xfree}(p \xrightarrow{a} S) = X$, where X is a X-free set for $p \xrightarrow{a} S$ such that the value of $need\mathsf{X}(p \xrightarrow{a} S, X)$ is minimal.

It is a straightforward observation that this function satisfies $\varphi^{\mathsf{Xfree}}(A) \in \mathrm{LTL}(\mathsf{U}^{lh(A)}, \mathsf{X}^{\mathsf{X}h(A)})$. ■

We should note that the bound on U*-depth*$(\varphi^{\mathsf{Xfree}}(A))$ given by $lh(A)$ is not tight. The structure of a formula φ'_p shows that there can be states with a loop that are translated into $\top$ or $\bot$ and thus they do not bring any

new temporal operators. However, if we remove these states and all transitions of the form $p \xrightarrow{a} S$ such that S contains a state translated into $\bot$, we get an automaton A' that is language equivalent to the original one and U-*depth*$(\varphi^{\mathsf{Xfree}}(A')) = lh(A')$.

Theorem 7.7 *Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ be a formula. Then $lh(A^\Sigma(\varphi)) \leq m$ and $\mathsf{X}h(A^\Sigma(\varphi)) \leq n$ for each alphabet Σ.*

Proof: Before we prove the inequalities, we examine the automaton $A^\Sigma(\varphi)$. Let $q_{\varphi'}$ be a state of $A^\Sigma(\varphi)$. States in $Succ(q_{\varphi'})$ are of the form

1. $q_{\psi \mathsf{U} \rho}$ or $q_{\neg(\psi \mathsf{U} \rho)}$, where $\psi \,\mathsf{U}\, \rho$ is such a subformula of φ' that is not in scope of any X operator, or

2. q_ψ or $q_{\neg\psi}$, where $\mathsf{X}\psi$ is such a subformula of φ' that is not in scope of any other X operator.

Let us note that some of the states can match both cases, e.g. a state $q_{\psi \mathsf{U} \rho} \in Succ(q_{(\psi \mathsf{U} \rho) \vee \mathsf{X}(\psi \mathsf{U} \rho)})$. Moreover, the definition of a transition function δ implies that only the states of the form $q_{\psi \mathsf{U} \rho}$ or $q_{\neg(\psi \mathsf{U} \rho)}$ can have a loop.

From the above observations and the definition of loop-height it directly follows that $lh(A^\Sigma(\varphi)) \leq \mathsf{U}$-*depth*$(\varphi) \leq m$.

Let $q_{\varphi'}$ be a state of the automaton and $Y \subseteq Succ'(q_{\varphi'})$ be a set of its successors of the first form (excluding these of both forms). In order to prove the second inequality of the theorem, we show that for every transition $q_{\varphi'} \xrightarrow{a} S$ the set $X_S = S \cap Y$ is X-free. The construction of an automaton $A^\Sigma(\varphi)$ implies that if a positive boolean formula $\delta(q_{\varphi'}, a)$ contains a state $q_{\psi \mathsf{U} \rho} \in Y$ then the state always occurs in the formula

$$\delta(q_{\psi \mathsf{U} \rho}, a) \;=\; \delta(q_\rho, a) \vee (\delta(q_\psi, a) \wedge q_{\psi \mathsf{U} \rho}).$$

Similarly, if $\delta(q_{\varphi'}, a)$ contains a state $q_{\neg(\psi \mathsf{U} \rho)} \in Y$ then the state always occurs in the formula

$$\overline{\delta(q_{\psi \mathsf{U} \rho}, a)} \;=\; \overline{\delta(q_\rho, a)} \wedge (\overline{\delta(q_\psi, a)} \vee q_{\neg(\psi \mathsf{U} \rho)}).$$

We show that each set X_S satisfies the conditions for X-freeness for every transition $q_{\varphi'} \xrightarrow{a} S$.

1. Let $q_{\psi \mathsf{U} \rho} \in X_S$. The property of $\delta(q_{\varphi'}, a)$ mentioned above gives us that there is a set $S' \subseteq S$ such that $S' \models \delta(q_\rho, a)$ or $S' \models \delta(q_\psi, a) \wedge q_{\psi \mathsf{U} \rho}$. Anyway, $q_{\psi \mathsf{U} \rho} \xrightarrow{a} S'$. The argumentation for the case $q_{\neg(\psi \mathsf{U} \rho)} \in X_S$ is similar.

2. Let $Y \subseteq X_S$ be such a set that for every $q \in Y$ there is a transition $q \xrightarrow{a} S'_q$ such that $q \notin S'_q$. Thus if q is of the form $q_{\psi \mathsf{U} \rho}$, then $S'_q \models \delta(q_\rho, a)$. Otherwise, q is of the form $q_{\neg(\psi \mathsf{U} \rho)}$ and $S'_q \models \overline{\delta(q_\rho, a)} \wedge \overline{\delta(q_\psi, a)}$. Again, the property of $\delta(q_{\varphi'}, a)$ mentioned above (together with the fact that $\delta(q_{\varphi'}, a)$ is in positive normal form) implies that there is a set $S'' \subseteq (S \setminus Y) \cup \bigcup_{q \in Y} S'_q$ satisfying $S'' \models \delta(q_{\varphi'}, a)$.

Proving the X-freeness of the considered sets we have demonstrated that the improved translation allows to omit the X operators in front of the subformulae corresponding to the successors of $q_{\varphi'}$ of the first form (and not of both forms). Let us recall that these successors correspond to the subformulae of the form $\psi \mathsf{U} \rho$ of the original formula φ' that are never in scope of any X operator in φ'. Hence, the improved translation with use of the Xfree function assigning the X-free sets defined above produces the formula $\varphi^{\mathsf{Xfree}}(A^\Sigma(\varphi))$ with at most the same X-*depth* as the original formula φ. We are done as $\mathsf{X}h(A^\Sigma(\varphi))$ keeps the lowest X-*depth* achievable by the improved translation and thus $\mathsf{X}h(A^\Sigma(\varphi)) \leq \mathsf{X}\text{-}depth(\varphi) \leq n$. ■

Combining Theorem 7.6 and Theorem 7.7 we get the following two corollaries.

Corollary 7.8 *For each formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ and each alphabet Σ there exists a function* Xfree *such that $\varphi^{\mathsf{Xfree}}(A^\Sigma(\varphi)) \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.*

Corollary 7.9 *For each A1W automaton $A = (\Sigma, Q, q_0, \delta, F)$ there exists a function* Xfree *such that*

$$lh(A) \geq lh(A^\Sigma(\varphi^{\mathsf{Xfree}}(A))) \text{ and } \mathsf{X}h(A) \geq \mathsf{X}h(A^\Sigma(\varphi^{\mathsf{Xfree}}(A))).$$

7.3 Defining LTL fragments via A1W automata

In this section we define classes of A1W automata matching fragments of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k)$, where $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$, and LTL fragments from the *until-release hierarchy* [ČP03]. All these fragments are given by syntactic constraints on LTL formulae. Basically, the classes of A1W automata can be defined by constraints on transition function and by constraints on the set of accepting states.

In order to improve the presentation of the following results, we overload the notation of LTL fragments; we identify an LTL fragment $\mathcal{F}$ with a set

$$\{L^\Sigma(\varphi) \mid \varphi \in \mathcal{F} \text{ and } \Sigma \text{ is an alphabet}\},$$

i.e. with a set of languages defined by formulae from the fragment. The interpretation of $\mathcal{F}$ is always clearly determined by the context.

7.3.1 Fragments $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k)$

In order to identify classes of A1W automata matching all LTL fragments of the form $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k)$ where $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$, we need to replace the bound on U-*depth*$(\varphi^{\mathsf{Xfree}}(A))$ given by the loop-height of A by bounds on U-*depth* and F-*depth* of the formula. Therefore we define another metrics called U-*height*. The definition of U-height reflects the structure of φ'_p.

Definition 7.10 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For each state $p \in Q$ we inductively define its* U-height, *written $\mathsf{U}h(p)$, as*

$$\mathsf{U}h(p) = \begin{cases} \max\{\mathsf{U}h(q) \mid q \in \mathit{Succ}'(p)\} + 1 & \text{if } p \in \mathit{Succ}(p) \text{ and} \\ & p \text{ is not of } \mathsf{F}\text{-type or } \mathsf{G}\text{-type}, \\ \max\{\mathsf{U}h(q) \mid q \in \mathit{Succ}'(p)\} & \text{otherwise}, \end{cases}$$

where maximum over empty set is 0. The U-height *of the automaton A is then defined as the* U-height *of its initial state, i.e. $\mathsf{U}h(A) = \mathsf{U}h(q_0)$.*

We are now ready to define the classes of A1W automata matching the considered LTL fragments. To shorten our notation, a class is formally defined as a set of languages accepted by A1W automata rather than a set of A1W automata.

Definition 7.11 *Let $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$. We define $\mathrm{A1W}(m, n, k)$ to be the set $\{L(A) \mid A$ is an A1W automaton and $\mathsf{U}h(A) \leq m$, $\mathsf{X}h(A) \leq n$, $lh(A) \leq m+k\}$.*

Theorem 7.12 *For all $m, n, k \in \mathbb{N}_0 \cup \{\infty\}$ it holds that $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k) = \mathrm{A1W}(m, n, k)$.*

Proof: Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n, \mathsf{F}^k)$ and Σ be an alphabet. In the LTL$\to$A1W translation every subformula $\mathsf{F}\psi$ is handled as $\top\,\mathsf{U}\,\psi$. Theorem 7.7 implies that $\mathsf{X}h(A^\Sigma(\varphi)) \leq n$ and $lh(A^\Sigma(\varphi)) \leq m + k$. Further, every state of the automaton $A^\Sigma(\varphi)$ corresponding to a subformula $\mathsf{F}\psi$ has the form $q_{\top\mathsf{U}\psi}$ or $q_{\neg(\top\mathsf{U}\psi)}$. One can readily check that each state $q_{\top\mathsf{U}\psi}$ is of F-type and each state $q_{\neg(\top\mathsf{U}\psi)}$ is of G-type. Hence, these states do not increase the U-height of the automaton. We get $\mathsf{U}h(A^\Sigma(\varphi)) \leq m$ and thus $L^\Sigma(\varphi) \in \mathrm{A1W}(m, n, k)$.

Let A be an A1W automaton such that $\mathsf{U}h(A) \leq m$, $\mathsf{X}h(A) \leq n$, and $lh(A) \leq m + k$. Theorem 7.6 says that the automaton can be translated into formula from $\mathrm{LTL}(\mathsf{U}^{m+k}, \mathsf{X}^n)$. As the definition of U-height follows the improved translation it is easy to check that $\varphi^{\mathsf{Xfree}}(A) \in \mathrm{LTL}(U^{\mathsf{U}h(A)}, X^n, \mathsf{F}^{lh(A)})$. Moreover, explicit treatment of F operator has no influence on the fact that arbitrary occurrence of F operator can be replaced by U operator. Hence, the automaton can be translated into a formula from $\mathrm{LTL}(U^m, X^n, \mathsf{F}^{lh(A)-m})$. Hence, $L(A) \in \mathrm{LTL}(U^m, X^n, \mathsf{F}^k)$. ■

Let us emphasize that Lemma 7.12 covers some distinguished LTL fragments. For example, languages defined by an $\mathrm{LTL}(\mathsf{U}^k, \mathsf{X}, \mathsf{F})$ fragment (fragments of this form constitute the *until hierarchy* [EW00, TW01]) can be defined by A1W automata with U-height at most k and vice versa. In particular, a language can be expressed by a formula from the $\mathrm{LTL}(\mathsf{X}, \mathsf{F})$ fragment (also called *restricted temporal logic* [PP04]) if and only if it is recognized by an A1W automaton such that every state with a loop is of F-type or G-type.

7.3.2 Until-release hierarchy

The hierarchy consists of fragments UR_i, RU_i for all i. The definition of the fragments does not care about X operators as well as the different expressiveness of F and U operators. Therefore we employ the standard translations given in Section 7.1.

Intuitively, we show that alternation of U and R operators in a formula corresponds to the alternation of nonaccepting and accepting states in the structure of an A1W automaton.

Definition 7.13 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For each $i \in \mathbb{N}_0$ we inductively define sets of states U_i and R_i as follows.*

- $U_0 = R_0 = \{p \mid lh(p) = 0\}$.
- *U_{i+1} is the smallest set of states satisfying*
  - *$U_i \cup R_i \subseteq U_{i+1}$ and*
  - *if $p \notin F$ and $Succ(p) \subseteq U_{i+1}$ then $p \in U_{i+1}$.*
- *R_{i+1} is the smallest set of states satisfying*
  - *$U_i \cup R_i \subseteq R_{i+1}$ and*
  - *if $p \in F$ and $Succ(p) \subseteq R_{i+1}$ then $p \in R_{i+1}$.*

We also define functions $U_A, R_A : Q \longrightarrow \mathbb{N}_0$ as

$$U_A(p) = min\{i \mid p \in U_i\} \text{ and } R_A(p) = min\{i \mid p \in R_i\}.$$

Definition 7.14 *For each $i \in \mathbb{N}_0$ we define sets $\mathcal{U}_i$ and $\mathcal{R}_i$ as*

$$\begin{aligned} \mathcal{U}_i &= \{L(A) \mid A = (\Sigma, Q, q_0, \delta, F) \text{ is an A1W automaton and } U_A(q_0) \leq i\}, \\ \mathcal{R}_i &= \{L(A) \mid A = (\Sigma, Q, q_0, \delta, F) \text{ is an A1W automaton and } R_A(q_0) \leq i\}. \end{aligned}$$

The classes $\mathcal{U}_i$ and $\mathcal{R}_i$ match the classes UR_i and RU_i, respectively. Before we give an explicit proof, we present some auxiliary results.

Definition 7.15 *Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. For each $i \in \mathbb{N}_0$ we inductively define sets of states U'_i and R'_i as follows.*

- $U'_0 = R'_0 = \{p \mid lh(p) = 0\}$.
- U'_{i+1} *is the smallest set of states satisfying*
  - $U'_i \cup R'_i \subseteq U'_{i+1}$ *and*
  - *if* p *has no loop or* $p \notin F$, *and* $Succ'(p) \subseteq U'_{i+1}$ *then* $p \in U'_{i+1}$.
- R'_{i+1} *is the smallest set of states satisfying*
  - $U'_i \cup R'_i \subseteq R'_{i+1}$ *and*
  - *if* p *has no loop or* $p \in F$, *and* $Succ'(p) \subseteq R'_{i+1}$ *then* $p \in R'_{i+1}$.

We also define functions $U'_A, R'_A : Q \longrightarrow \mathbb{N}_0$ *as*

$$U'_A(p) = min\{i \mid p \in U'_i\} \text{ and } R'_A(p) = min\{i \mid p \in R'_i\}.$$

Lemma 7.16 *For every A1W automaton* A *with an initial state* q_0 *there exists an A1W automaton* B *with an initial state* q'_0 *such that* $U'_A(q_0) = U_B(q'_0)$ *and* $L(A) = L(B)$.

Proof: On intuitive level, U_i counts the maximal alternation of nonaccepting and accepting states of the automaton, while U'_i counts just the alternation of nonaccepting and accepting states with a loop. Hence, we need to modify an automaton A is such a way that the states without any loop do not increase the number of alternations of nonaccepting and accepting states. To do this, we make an accepting and a nonaccepting copy of each state without any loop and we modify transition function such that every state without any loop in every transition is replaced by one of its two copies. As acceptance or nonacceptance of states without any loop has no influence on language given by the automaton, the modified automaton accepts the same language as the original one.

Let $A = (\Sigma, Q, q_0, \delta, F)$ be an A1W automaton. Let W denote the set of its states without any loop. We set $B = (\Sigma, Q', q'_0, \delta', F')$, where

- $Q' = (Q \setminus W) \cup \{q^a, q^n \mid q \in W\}$,
- $q'_0 = q_0$ if q_0 has a loop, $q'_0 = q_0^n$ otherwise,
- $F' = (F \setminus W) \cup \{q^a \mid q \in W\}$.

For every $p \in Q'$, by $o(p)$ we denote a state of the original system corresponding to the state p:

$$o(p) = \begin{cases} q & \text{if } p = q^a \text{ or } p = q^n \\ p & \text{otherwise} \end{cases}$$

For every $p \in Q'$ and $a \in \Sigma$, the positive boolean formula $\delta'(p, a)$ arises from $\delta(o(p), a)$ by replacement of every state $q \in W$ with

- q^a if $p \in F'$, or
- q^n otherwise.

It remains to show that $U'_A(q_0) = U_B(q'_0)$. From the construction of the automaton B it follows that $U'_B(p) = U'_A(o(p))$ for every $p \in Q'$. As $o(q'_0) = q_0$, it is sufficient to show that $U'_B(q'_0) = U_B(q'_0)$. If $lh(q'_0) = 0$, then we are done as $U'_B(q'_0) = 0 = U_B(q'_0)$. In the rest of the proof we show that the equation holds for $lh(q'_0) > 0$ as well.

For each state $p \in Q'$ we define $Succ^*(p)$ to be a transitive closure of $Succ'$ relation, i.e. $Succ^*(p)$ is the smallest set satisfying

- $Succ'(p) \subseteq Succ^*(p)$ and
- if $q \in Succ^*(p)$ then $Succ'(q) \subseteq Succ^*(p)$.

Further, by $LSucc^*(p)$ we denote the set of all states with a loop that are in $Succ^*(p)$.

From the construction of the automaton B it follows that for every state $p \in Q'$ such that $lh(p) > 0$ the following equations hold. Again, maximum over empty set is 0.

$$U_B(p) = \begin{cases} \max\{R_B(q) \mid q \in LSucc^*(p) \cap F'\} + 1 & \text{if } p \notin F' \\ R_B(p) + 1 & \text{if } p \in F' \end{cases}$$

$$\pi_B(p) = \begin{cases} U_B(p) + 1 & \text{if } p \notin F' \\ \max\{U_B(q) \mid q \in LSucc^*(p) \setminus F'\} + 1 & \text{if } p \in F' \end{cases}$$

If we replace in the above equations the function U_B with U'_B and the function R_B with R'_B, the resulting equations hold for each state p with a loop. Hence, for every state $p \in Q'$ with a loop it holds that $U'_B(p) = U_B(p)$ and $R'_B(p) = R_B(p)$. In particular, if q'_0 has a loop then $U'_B(q'_0) = U_B(q'_0)$.

Let us note that if q'_0 has no loop then it is a nonaccepting state. Further, one can prove that if $lh(q'_0) > 0$, q'_0 has no loop, and $LSucc^*(q'_0) \cap F' = \emptyset$ then $U'_B(q'_0) = 1 = U_B(q'_0)$.

Finally, let us assume that $lh(q'_0) > 0$, q'_0 has no loop, and $LSucc^*(q'_0) \cap F' \neq \emptyset$. As $q'_0 \notin F'$, then

$$\begin{aligned} U'_B(q'_0) &= \max\{U'_B(q) \mid q \in LSucc^*(q'_0)\} && (7.1)\\ &= \max\{U_B(q) \mid q \in LSucc^*(q'_0)\} && (7.2)\\ &\geq \max\{U_B(q) \mid q \in LSucc^*(q'_0) \cap F'\} && (7.3)\\ &= \max\{R_B(q) + 1 \mid q \in LSucc^*(q'_0) \cap F'\} && (7.4)\\ &= \max\{R_B(q) \mid q \in LSucc^*(q'_0) \cap F'\} + 1 && (7.5)\\ &= U_B(q'_0), && (7.6) \end{aligned}$$

where (7.1) follows from the definition of U'_B, (7.2) is due to the fact that for states with a loop the functions U'_B and U_B coincide, and the equation $U_B(q) = R_B(q) + 1$ valid for all $q \in F'$ gives us (7.4). To sum up, $U'_B(q'_0) \geq U_B(q'_0)$. We are done as it is easy to see that $U'_B(p) \leq U_B(p)$ holds for each state p of an arbitrary A1W automaton B. ■

By analogy, for an A1W automaton A with initial state q_0 one can construct a language equivalent A1W automaton B with an initial state q'_0 such that $R'_A(q_0) = R_B(q'_0)$.

Theorem 7.17 *For each $i \in \mathbb{N}_0$ it holds that $\mathsf{UR}_i = \mathcal{U}_i$ and $\mathsf{RU}_i = \mathcal{R}_i$.*

Proof: We focus on the former equation as the proof of the latter one is analogous. In order to prove the inclusion $\mathsf{UR}_i \subseteq \mathcal{U}_i$, we show that for every alphabet Σ and a formula $\varphi \in \mathsf{UR}_i$ the automaton $A = A^\Sigma(\varphi)$ given by standard LTL→A1W translation satisfies $U'_A(q_\varphi) \leq i$, where q_φ is an initial state of the automaton. This is sufficient due to Lemma 7.16.

Please note that operators U and R are not in scope of any negation in φ. Hence, the states of the automaton $A^\Sigma(\varphi)$ can be divided into three kinds according to the corresponding subformula of φ.

1. A subformula of the form $\mathsf{X}(\psi \,\mathsf{U}\, \rho)$ or $\psi \,\mathsf{U}\, \rho$ is translated into a state $q_{\psi \mathsf{U} \rho}$ satisfying $q_{\psi \mathsf{U} \rho} \notin F$.

2. As $\psi \,\mathsf{R}\, \rho$ is seen as an abbreviation for $\neg(\neg\psi \,\mathsf{U}\, \neg\rho)$, a subformula of the form $\mathsf{X}(\psi \,\mathsf{R}\, \rho)$ or $\psi \,\mathsf{R}\, \rho$ is translated into a state $q_{\neg(\neg\psi \mathsf{U} \neg\rho)} \in F$.

3. A subformula of the form $\mathsf{X}\psi$ that is not covered by the previous cases is translated into a state q_ψ such that $q_\psi \notin F$ and q_ψ has no loop.

To sum up, states corresponding to U operator are not accepting while the states corresponding to R operator are accepting. Moreover, states corresponding to U or R are the only states that can have a loop. From this observation and from the structure of the transition function δ it follows that each subformula ψ of φ satisfies

$$\psi \in \mathsf{UR}_j \implies U'_A(q_\psi) \leq j \;\text{ and }\; \psi \in \mathsf{RU}_j \implies R'_A(q_\psi) \leq j.$$

In particular, $U'_A(q_\varphi) \leq i$.

To prove the inclusion $\mathsf{UR}_i \supseteq \mathcal{U}_i$ we assume that A is an A1W automaton with an initial state q_0 satisfying $U_A(q_0) \leq i$. We show that the formula $\varphi(A)$ given by the standard A1W→LTL translation can be equivalently expressed by a formula from $\mathsf{UR}_{U_A(q_0)}$.

Here we employ the fact that the formulae $(\sigma \,\mathsf{U}\, \xi) \wedge \mathsf{G}\sigma$ and $((\mathsf{X}\xi) \,\mathsf{R}\, \sigma) \vee \xi$ are equivalent for all subformulae σ, ξ. Therefore, for each state p of the

automaton A the formula φ_p can be equivalently given as follows:

$$\varphi_p = \begin{cases} \sigma_p \,\mathsf{U}\, \xi_p & \text{if } p \notin F \\ ((\mathsf{X}\xi_p)\,\mathsf{R}\,\sigma_p) \vee \xi_p & \text{if } p \in F \end{cases}$$

Using this modified construction, we get a formula that is equivalent to $\varphi(A)$. Moreover, U operators correspond to nonaccepting states while R operators correspond to the accepting ones. Hence, the alternation of nonaccepting and accepting states in the automaton correspond to the alternation of U and R operators in the resulting formula. In other words, the formula is in $\mathsf{UR}_{U_A(q_0)}$. ■

Theorem 7.17 and Corollary 3.51 give us the following corollary.

Corollary 7.18 *An ω-language L is definable by LTL if and only if $L \in \mathcal{U}_3 \cap \mathcal{R}_3$.*

7.4 Additional notes

Our research on deeper connections between A1W automata and LTL fragments brought several topics for future work. The most interesting topics follow.

One such a topic is a situation on finite words. We think that the improved translation works for alternating automata over finite words as well. However, we suppose that the connection between until-release hierarchy and alternation of nonaccepting and accepting states does not extend to finite words.

In Definition 7.1 we formulate two conditions which allow to decrease the X-*depth* of a formula corresponding to a given A1W automaton. As the conditions are quite complicated, it is only natural to ask whether there are some simpler (or more general) conditions with the same effect. For example, we have no counterexample showing that the improved translation produces incorrect results when we define $\mathsf{Xfree}(p \xrightarrow{a} S)$ to be a set of all states $q \in S \setminus \{p\}$ satisfying

1. there is $S' \subseteq S$ such that $q \xrightarrow{a} S'$, and
2. if $q \xrightarrow{a} S'$ and $q \notin S'$ then there exists $S'' \subseteq (S \setminus \{q\}) \cup S'$ such that $p \xrightarrow{a} S''$.

Unfortunately, we have no proof of correctness of the translation for this Xfree function.

Another question is a succinctness of A1W automata. Let us consider a formula φ with a more than one copy of a subformula ψ, e.g. $\varphi = (a \vee \mathsf{X}\psi)\,\mathsf{U}\,(b \wedge \mathsf{XX}\psi)$. All copies of the subformula correspond to one state q_ψ of the automaton produced by LTL$\to$A1W translation. Therefore we suppose

(but we have no proof yet) that an A1W automaton can be exponentially more succinct than arbitrary corresponding LTL formula in some cases.

The last topic we mention here is connected to automata rather than the logic. In this chapter we work with the alternating Büchi automata that are 1-weak. The 1-weakness can be generalized to k-weakness in the following way. An alternating Büchi automaton is called *weak* if the set of states Q can be partitioned into disjoint sets $Q_1, Q_2, \ldots, Q_n$ such that

- if $q \in Succ(p)$, $q \in Q_i$, and $p \in Q_j$ then $i \leq j$, and
- $Q_i \cap F = \emptyset$ or $Q_i \subseteq F$ for every $0 < i \leq n$,

where F is a set of accepting states. Moreover, the automaton is called k-*weak* if $|Q_i| \leq k$ for every $0 < i \leq n$. It is known that general alternating weak automata recognize all ω-regular languages, whereas alternating 1-weak automata recognize all ω-languages definable by LTL. The definition of alternating k-weak automata (or *AkW automata* for short) brings several interesting questions:

- What is the expressive power of AkW automata?
- Is the hierarchy of classes of AkW automata expressively strict?
- For each k, is there any natural extension LTL$_k$ of LTL such that AkW automata define the same languages as LTL$_k$?

Chapter 8

Conclusions

This thesis provides a wide overview of LTL fragment properties that are relevant to model checking. With use of this overview one can easily see the relations between expressiveness of LTL fragments and theoretical complexities of the model checking problem for these fragments. This thesis also covers several fragment properties that lead or can potentially lead to improvements of the model checking process (e.g. closure under stuttering leads to applicability of partial order reduction methods).

Further, this thesis contains original results divided into three areas: extended stuttering principles, characteristic patterns, and deeper connections between LTL and alternating 1-weak Büchi automata. Some applications of these results in model checking are suggested as well.

8.1 Future work

The sections called *Additional notes* and located at the end of each chapter include a number of interesting topics for future work. Some of these topics are open questions borrowed from the cited papers. In the near future the author plans to concentrate on the development of new model checking algorithms and methods employing some of the results presented in this thesis, namely the letter stuttering principle, the general stuttering principle, and the concept of characteristic patterns.

Bibliography

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, 1992.

[Arn85] André Arnold. A syntactical congruence for rational ω-languages. *Theoretical Computer Science*, 39:333–335, 1985.

[Bur74] Rod M. Burstall. Program proving as hand simulation with a little induction. In *Proceedings of International Congress of the International Federation for Information Processing*, pages 308–312. North-Holland, 1974.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

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

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[CH91] Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.

[CMP92] Edward Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer-Verlag, 1992.

[ČP03] Ivana Černá and Radek Pelánek. Relating hierarchy of temporal properties to model checking. In *Proceedings of the 30th Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, volume 2747 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[CPP93] Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pages 411–420. IEEE Computer Society Press, 1999.

[Dam99] Dennis R. Dams. Flat fragments of CTL and CTL*: Separating the expressive and distinguishing powers. *Logic Journal of the IGPL*, 7(1):55–78, 1999.

[DGV99] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer-Verlag, 1999.

[DS02] Stéphane Demri and Philippe Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–168, 2000.

[Eme90] E. Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 995–1072. Elsevier, 1990.

[Ete00] Kousha Etessami. A note on a question of Peled and Wilke on stutter-invariant LTL. *Information Processing Letters*, 75(6):261–263, 2000.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.

[EW00] Kousha Etessami and Thomas Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160:88–108, 2000.

[Gab89] Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Conference on Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 2001.

[God96] Patrice Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Conference Record of the 7th Annual ACM Symposium on Principles of Programming Languages (POPL'80)*, pages 163–173. ACM Press, 1980.

[GPVW95] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.

[HKP82] David Harel, Dexter Kozen, and Rohit Parikh. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25(2):144–170, 1982.

[Hol97] Gerard J. Holzmann. The Model Checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997. Special issue on Formal Methods in Software Practice.

[HP95] Gerard Holzmann and Doron Peled. Partial order reduction of the state space. In *Proceedings of the 1st SPIN Workshop (SPIN'95)*, 1995. A position paper.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Kam68] Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[Kri63] Saul A. Kripke. Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Krö87] Fred Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[KS02] Antonín Kučera and Jan Strejček. The stuttering principle revisited: On the expressiveness of nested X and U operators in the logic LTL. In J. Bradfield, editor, *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic (CSL'02)*, volume 2471 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 2002.

[KS04] Antonín Kučera and Jan Strejček. An effective characterization of properties definable by LTL formulae with a bounded nesting depth of the next-time operator. Technical Report FIMU-RS-2004-04, Faculty of Informatics, Masaryk University Brno, 2004.

[KS05a] Antonín Kučera and Jan Strejček. Characteristic patterns for LTL. In *Proceedings of SOFSEM'05*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.

[KS05b] Antonín Kučera and Jan Strejček. The stuttering principle revisited. *Acta Informatica*, 2005. To appear.

[Lad77] Richard E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.

[Lam83a] Leslie Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.

[Lam83b] Leslie Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proceedings of the IFIP Congress on Information Processing*, pages 657–667, Amsterdam, 1983. North-Holland.

[Lar94] François Laroussinie. *Logique temporelle avec passé pour la spécification et la vérification des systèmes réactifs*. PhD thesis, I.N.P. de Grenoble, 1994.

[LMS02] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Temporal logic with forgettable past. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 383–392. IEEE Computer Society Press, 2002.

[LP85] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages (POPL'85)*, pages 97–107. ACM Press, 1985.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In R. Parikh, editor, *Proceedings of the Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer-Verlag, 1985.

[LS95] François Laroussinie and Philippe Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.

[LT00] Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words (extended abstract). In J. van Leeuwen et al., editors, *Proceedings of the 1st IFIP International Conference on Theoretical Computer Science (IFIP TCS'00)*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer-Verlag, 2000.

[Mai00] Monika Maidl. The common fragment of CTL and LTL. In D. C. Young, editor, *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 643–652. IEEE Computer Society Press, 2000.

[Mar03a] Nicolas Markey. *Logiques temporelles pour la vérification: expressivité, complexité, algorithmes*. PhD thesis, University of Orléans, 2003.

[Mar03b] Nicolas Markey. Temporal logic with past is exponentially more succinct. *Bulletin of the European Association for Theoretical Computer Science*, 79:122–128, 2003.

[Mar04] Nicolas Markey. Past is for free: on the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6–7):431–458, 2004.

[McM99] Kenneth L. McMillan. Verification of infinite state systems by compositional model checking. In *Proceedings of the 10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 219–237. Springer-Verlag, 1999.

[Mey75] Albert R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathmatics*, pages 132–154. Springer-Verlag, 1975.

[MP71] Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, Cambridge, Mass., 1971.

[MP90a] Oded Maler and Amir Pnueli. Tight bounds on the complexity of cascaded decomposition of automata. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS'90)*, volume II, pages 672–682. IEEE Computer Society Press, 1990.

[MP90b] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC'90)*, pages 377–410. ACM Press, 1990.

[MP91] Zohar Manna and Amir Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[MP94] Oded Maler and Amir Pnueli. On the cascaded decomposition of automata, its complexity and its application to logic. Available at `<http://www-verimag.imag.fr/PEOPLE/Oded.Maler/Papers/decomp.ps>`, 1994.

[MS97] Oded Maler and Ludwig Staiger. On syntactic congruences for ω-languages. *Theoretical Computer Science*, 183(1):93–112, 1997.

[MS03] Nicolas Markey and Philippe Schnoebelen. Model checking a path (preliminary report). In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–265. Springer-Verlag, 2003.

[MSS88] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the 3rd Annual IEEE Symposium on Logic in Computer Science (LICS'88)*, pages 422–427. IEEE Computer Society Press, 1988.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PC03] Dimitrie O. Păun and Marsha Chechik. On closure under stuttering. *Formal Aspects of Computing*, 14:342–368, 2003.

[Pel98] Doron Peled. Ten years of partial order reduction. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 17–28. Springer-Verlag, 1998.

[Per84] Dominique Perrin. Recent results on automata and infinite words. In M. P. Chytil and V. Koubek, editors, *Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science (MFCS'84)*, volume 176 of *Lecture Notes in Computer Science*, pages 134–148. Springer-Verlag, 1984.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[PP86] Dominique Perrin and Jean-Eric Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32(3):393–406, 1986.

[PP04] Dominique Perrin and Jean-Eric Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[Pri57] Arthur N. Prior. *Time and Modality*. Oxford University Press, 1957.

[PS04] Radek Pelánek and Jan Strejček. Deeper connections between LTL and alternating automata. Technical Report FIMU-RS-2004-08, Faculty of Informatics, Masaryk University Brno, 2004.

[PW97a] Doron Peled and Thomas Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, 1997.

[PW97b] Jean-Eric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.

[PWW98] Doron Peled, Thomas Wilke, and Pierre Wolper. An algorithmic approach for checking closure properties of ω-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.

[Roh97] Scott Rohde. *Alternating automata and the temporal logic of ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[RP86] Roni Rosner and Amir Pnueli. A choppy logic. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 306–313. IEEE Computer Society Press, 1986.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer-Verlag, 2000.

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

[Sch65] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[Sch00] Heinz Schmitz. Restricted temporal logic and deterministic languages. *Journal of Automata, Languages and Combinatorics*, 4(3):325–342, 2000.

[Sch01] Klaus Schneider. Improving automata generation for linear temporal logic by considering the automaton hierarchy. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'01)*, volume 2250 of *Lecture Notes in Computer Science*, pages 39–54. Springer-Verlag, 2001.

[Sch03] Philippe Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic, vol. 4, selected papers from 4th Conf. Advances in Modal Logic (AiML'02)*, pages 437–459. King's College Publication, 2003.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Co., 1997.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, pages 1–9, 1973.

[Sto74] Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

[SV89] Shmuel Safra and Moshe Y. Vardi. On ω-automata and temporal logic. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC'89)*, pages 127–137. ACM Press, 1989.

[Tau03] Heikki Tauriainen. On translating linear temporal logic into alternating and nondeterministic automata. Research Report A83, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2003.

[Tho79] Wolfgang Thomas. Star-free regular sets of ω-sequences. *Information and Control*, 42(2):148–156, 1979.

[Tho81] Wolfgang Thomas. A combinatorial approach to the theory of ω-automata. *Information and Control*, 48(3):261–283, 1981.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 4, pages 133–191. Elsevier, 1990.

[Thu06] Axel Thue. Über unendliche Zeichenreihen. *Kra. Vidensk. Selsk. Skrifter, I. Mat. Nat. Kl.*, 1906(7):1–22, 1906.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation: $FO^2 = \Sigma_2 \cap \Pi_2$. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 234–240. ACM Press, 1998.

[TW01] Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. *SIAM Journal on Computing*, 31(3):777–798, 2001.

[TW02] Denis Thérien and Thomas Wilke. Nesting Until and Since in linear temporal logic. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 455–464. Springer-Verlag, 2002.

[Val91] Antti Valmari. A stubborn attack on state explosion. In E. M. Clarke and R. P. Kurshan, editors, *Proceedings of the 2nd International Conference on Computer Aided Verification (CAV'90)*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165. Springer-Verlag, 1991.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of*

the 1st Annual IEEE Symposium on Logic in Computer Science (LICS'86), pages 322–331. IEEE Computer Society Press, 1986.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Society for Industrial and Apllied Mathematics, 2000.

[Wil98] Thomas Wilke. Classifying discrete temporal properties. Habilitationsschrift (post-doctoral thesis), 1998.

[Wil99] Thomas Wilke. Classifying discrete temporal properties. In Chr. Meinel and S. Tison, editors, *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1999.

[Wol83] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths (extended abstract). In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Computer Society Press, 1983.

[Zha03] Wenhui Zhang. Combining static analysis and case-based search space partitioning for reducing peak memory in model checking. *Journal of Computer Science and Technology*, 18(6):762–770, 2003.

[Zuc86] Lenore D. Zuck. *Past Temporal Logic*. PhD thesis, Weizmann Institute, 1986.