Saarland University
Faculty of Mathematics and Computer Science
Department of Computer Science

Master's Thesis

Bounded Synthesis for Past LTL

submitted by
Peter Wita

submitted on
September 26, 2018

Supervisor
Prof. Bernd Finkbeiner, Ph.D.

Advisor
Felix Klein, M.Sc.

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

Eidesstattliche Erklärung

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

Statement in Lieu of an Oath

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

Einverständniserklärung

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

Declaration of Consent

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

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Linear-time Temporal Logic (LTL) is a popular specification language to describe the input and output behavior of a system. Its synthesis problem aims for the automatic derivation of an implementation that meets an input LTL formula. To traverse this enormous (and possibly infinite) search space of candidates in a structured manner, Finkbeiner and Schewe introduced the idea of bounded synthesis. Here a given LTL specification is first translated into a universal co-Büchi tree automaton before an implementation of bounded size is guessed and encoded within a constraint system.

Past Linear-time Temporal Logic (PLTL) characterizes an extension of LTL, which allows to refer to the past by using additional modalities. A resistance to its inclusion was based on the request of minimality and Gabbay's Separation Theorem that proves PLTL to add no expressive power by providing a rewriting algorithm of non-elementary complexity. However, recent research has proven that it is more succinct than pure-future LTL. Furthermore, the complexity of its verification problem is still PSPACE-complete.

In this thesis, we consider the bounded synthesis problem for PLTL. We present three solutions, each focusing on a different part of the original approach. The first one considers Gabbay's Separation Theorem and, additionally, presents a complete rewriting of its algorithm. The others describe a partial approach using temporal testers as well as an approach using an algorithm from PLTL to non-deterministic Büchi automata. To evaluate our approaches we additionally compare their efficiency for the famous future(past)-fragment of PLTL.

Acknowledgement

I am very grateful to Prof. Bernd Finkbeiner for offering me this interesting and challenging topic. Moreover, I would like to thank Prof. Bernd Finkbeiner and Prof. Holger Hermanns for reviewing this thesis. I also want to thank my advisor Felix Klein for the guidance during the development giving me a valuable support over the recent months.

Furthermore, I want to give a warm thanks to my fellow student and good friend Dominic Buchheit, as well as to Gerdi Maurer-Landwehr for proof-reading the thesis.

Finally, I want to thank my family for their support and that they have always believed in me. We have made it!

Contents

Chapter 1

Introduction

Programs that are continuously interacting with their environment are known as *reactive systems*. Whenever the environment gives some input to the system, e.g., a request, the system has to act accordingly by giving a suitable reaction as output, e.g., a grant.

To describe properties and behaviors of reactive systems Amir Pnueli introduced the specification language *Linear-time Temporal Logic* [23], which became very popular due to its intuitive operators to make statements referring to the future of a system state. Given an LTL specification and an implementation for a reactive system, the LTL verification problem checks whether the implementation satisfies the desired property and presents a counterexample in the negative case. Sistla and Clark showed that its complexity is PSPACE-complete [27].

Instead of writing his own code and having to debug upcoming errors manually during the model checking, Church identified the synthesis problem [6]. Given only an LTL specification, LTL synthesis manually derives an implementation for a reactive system. Besides the huge complexity of the synthesis problem, which is proven to be 2EXPTIME-complete [16, 25], its main drawback is the enormous search space of implementations satisfying the input specification. To traverse this search space in a structured manner Finkbeiner and Schewe introduced the approach of bounded synthesis [9]. Starting with the transformation of an LTL formula into a universal-co Büchi tree automaton, they showed that the acceptance of an implementation by the automaton is equivalent to an annotation that maps each pair of states to a natural number. Thus, keeping the mapping below an upper bound, which mainly depends on the system parameters, e.g., the rejecting states, their approach allows them to observe the simplest solutions from the beginning onwards. Additionally, they showed a reduction from the co-Büchi winning condition to a Safety game. Finally, they made another reduction by specifying an implementation of bounded size within a constraint-system as input for an SMT solver.

Other approaches to solve the synthesis problem focused on various fragments of full LTL [1, 2] mainly describing a solution which uses 2-player games. Bloem et al. observed so-called GR(1)-games [4] whose winning condition is described by the GR(1)-fragment of LTL. It requires the system to satisfy its set of guarantees whenever the environment meets its assumptions. They derived a symbolic algorithm for solving GR(1)-games, which could be used to solve the synthesis of the GR(1)-fragment. As a result, they further made observations on the past-extension of standard LTL, known as *Past Linear-time Temporal Logic* (PLTL). Adding pure-past formulas to the sets of assumptions and guarantees, respectively, allows them a symbolic solution for the synthesis problem of the described PLTL fragment. To do so, they build an encoding of the inner PLTL subformulas using *temporal testers* that keep track of their satisfiability.

PLTL was introduced in Kamp's thesis [14] as an extension of standard LTL by a set of operators observing the past. A lot of research resisted its inclusion based on the pursuit of minimality and the observations of *Gabbay's Separation Theorem* [10], where its equal expressiveness for the restriction of system executions to a definite start point in time is proven. On the other hand, PLTL provides an easier and more natural formulation of specifications in many interesting cases [20]. Consider as an example the PLTL specification in Equation 1.1, which requires the system to only output a grant g whenever it received a request r in the past, using the operators $\Box$ ("Always") and $\diamondsuit$ ("Once").

$$\varphi' = \Box(g \rightarrow \diamondsuit r) \tag{1.1}$$

Equation 1.2 gives an equivalent and one of the simplest representations using standard LTL with the operator $\mathcal{U}$ ("Until").

$$\varphi = \neg\big(\neg r \,\mathcal{U}\, (g \wedge \neg r)\big) \tag{1.2}$$

Clearly, the second formula is much harder to read and understand. Furthermore, Laroussinie et al. proved PLTL to be more succinct than standard LTL [17].

Although the PLTL verification problem is PSPACE-complete [27], its implementation is more difficult and complex in practice [19]. Thus, some research focused on its bounded variant [3, 19, 24], where a counterexample is only searched for finitely representable paths. Cimatti et al. [7] further extended this idea by only considering fragments of PLTL. They assumed an equivalent *Separated Normal Form* for their input formula that is part of the GR(1)-fragment and only allows pure-past formulas in its assumptions. Similar to [4], they took the truth values of subformulas into account when building a symbolic representation via *observer automata*. Both ideas are based on a PLTL equivalent construct using *history variables* that keep track of the previous states of the execution of a reactive system [20].

In this thesis, we focus on the bounded synthesis approach for PLTL extending the idea of [9]. We present three approaches, each operating on a different part of the original work. The first one, in the following called *naive approach*, operates on formula-level by applying Gabbay's Separation Theorem on the input formula. The second approach, referred to as *allrounder*, focuses on the creation of a universal co-Büchi tree automaton for the given PLTL formula. It is based on the approach of Gastin and Oddoux [12], who take a detour to two-way automata. Finally, our third approach follows the ideas of [7, 20] by observing bounded synthesis for a fragment of PLTL, giving it its denotation as *partial approach*. Rather than considering the GR(1)-fragment, we specify on a super set, known as *future(past)-fragment*, where a set of pure-past formulas is only combined with boolean and future connectives. The inner past formulas are encoded as temporal testers which are integrated into the annotation search. Afterwards, we present a comparison between partial approach and allrounder, which states their main drawbacks and advantages.

This thesis is structured as follows: Chapter 2 presents important preliminaries in order to understand the upcoming approaches. The three approaches are presented in the succeeding chapters, starting with the naive approach (Chapter 3), followed by the allrounder (Chapter 4) and the partial approach (Chapter 5). Finally, a comparison between the approaches is presented in Chapter 6, before giving a conclusion and an outlook on possible future work in Chapter 7.

Chapter 2

Preliminaries

In this chapter, we present previous work concerning our upcoming approaches. After introducing some notations and the specification language Linear-time Temporal Logic (LTL) along with its past extension (PLTL), we present the description of implementations in various ways. Subsequently, we talk about word and tree automata, followed by their properties and acceptance conditions. Finally, we present a construction transforming PLTL to Büchi word automata (BW), and introduce the idea of bounded synthesis for LTL. Figure 2.1 gives a rough idea of the various topics presented in this section and the way they are related with each other connected.

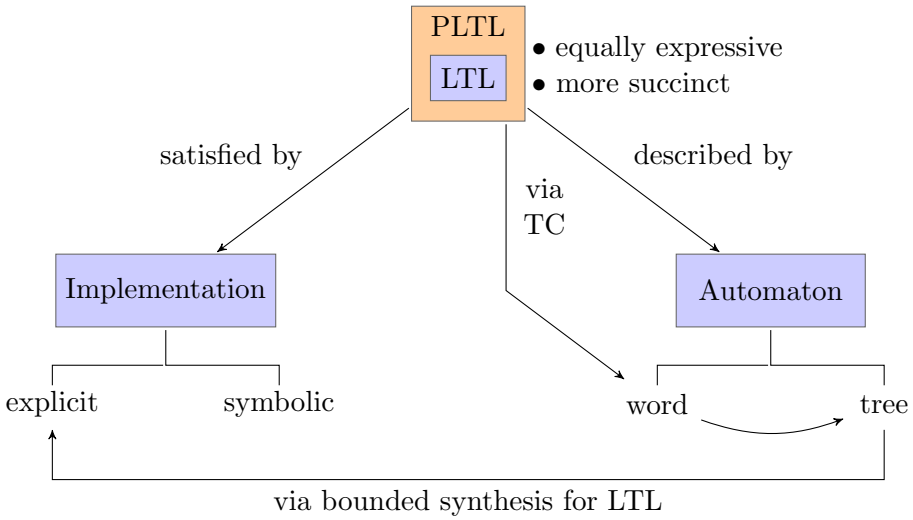


Figure 2.1: Overview of the presented topics of Chapter 2. TC stands for "Tableau Construction".

2.1 Notations

The set of *natural numbers* is denoted by $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of *positive natural numbers* by $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$, and the set additionally including -1 by $\mathbb{N}^{-1} = \mathbb{N} \cup \{-1\}$. Given two natural numbers $n, m \in \mathbb{N}$ with $n \leq m$, the interval $[n, m]$ denotes the set of all numbers between n and m, i.e., $\{n, n+1, \ldots, m\}$. The special case $\{0, 1, \ldots, n-1\}$ is denoted by $[n]$.

An *alphabet* Σ describes a non-empty and finite set of symbols, so called *letters*, where $|\Sigma|$ describes the number of symbols that it contains. Given an alphabet Σ, the concatenation $\alpha = \alpha_0\alpha_1 \ldots \alpha_{n-1}$ of finitely many letters of Σ is called a *finite word* over Σ, where $|\alpha| = n$ denotes the *length* of α. Another important non-empty and finite set of symbols is the *set of directions*, usually denoted by Υ. In here, the symbols are called *directions*. The concatenation $\alpha = \alpha_0\alpha_1 \ldots \alpha_{n-1}$ of finitely many directions of Υ is called a *finite branch* over Υ, where $|\alpha| = n$ denotes the *depth* of α. The symbol ϵ either describes the *empty word* or the *empty branch* with $|\epsilon| = 0$, respectively, based on context. The set of all finite words over Σ or all finite branches over Υ is denoted by Σ^* or Υ^*, respectively. Given some word or branch α and some position $n \in [|\alpha|]$, the n-th letter or n-th direction of α is denoted by α_n, starting with the 0-th letter or direction, respectively.

In the following, we talk about infinite sequences using the ω-symbol. By building a concatenation of infinitely many letters or directions, we observe *infinite words* or *branches* of infinite length or depth, respectively. The set Σ^ω defines the set of all infinite words over the alphabet Σ, Υ^ω the set of all infinite branches over the set of directions Υ. Additionally, the union of all finite and infinite branches is abbreviated by $\Upsilon^\infty = \Upsilon^* \cup \Upsilon^\omega$. Given an alphabet Σ, each subset of Σ^* defines a *language over finite words*, whereas each subset of Σ^ω defines a *language over infinite words*, called ω-language.

2.2 Past Linear-time Temporal Logic (PLTL)

LTL was introduced to describe properties of reactive systems. It is proven that its extension to *Past Linear-time Temporal Logic* (PLTL) is more succinct [21], as well as arguably more intuitive for many specifications. We use its definition as presented in [19], as well as its symbolic notation used, e.g., in [4].

2.2.1 Syntax

The syntax of PLTL is defined over a set of *atomic propositions* AP, including *true*, *false*. Given a PLTL formula φ over AP, the notation $AP_\varphi \subseteq AP$ denotes the set of all atomic propositions used in φ. Furthermore, the PLTL syntax consists of the standard boolean connectives $\neg$ ("Negation"), and $\vee$ ("Disjunction"). As time-concerning connectives, we present the future

operators $\bigcirc$ ("Next"), and $\mathcal{U}$ ("Until"), as well as the past operators $\ominus$ ("Past"), and $\mathcal{S}$ ("Since"). Thus, a PLTL formula φ is generated by the following grammar:

$$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \ominus\varphi \mid \varphi\,\mathcal{S}\,\varphi.$$

We talk about a boolean formula or assertion whenever there are no temporal connectives in our formula. An LTL formula is a formula without past-concerning operators, also called *pure-future*. Otherwise, we talk about a PLTL formula. A PLTL formula is called *pure-past* if it is build by boolean and past-connectives only.

Furthermore, there are multiple abbreviations within our grammar. Consider some arbitrary PLTL formulas φ_1 and φ_2. For the boolean connectives, we distinguish the abbreviations $\wedge$ ("Conjunction"), $\rightarrow$ ("Implication"), and $\leftrightarrow$ ("Equivalence"):

- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

The abbreviations for the future connectives are distinguished by $\mathcal{R}$ ("Release"), and the unary connectives $\Diamond$ ("Eventually") and $\Box$ ("Always"):

- $\varphi_1\,\mathcal{R}\,\varphi_2 \equiv \neg(\neg\varphi_1\,\mathcal{U}\,\neg\varphi_2)$
- $\Diamond\varphi_1 \equiv \mathit{true}\,\mathcal{U}\,\varphi_1$
- $\Box\varphi_1 \equiv \neg\Diamond\neg\varphi_1 \equiv \mathit{false}\,\mathcal{R}\,\varphi_1$

The past abbreviations are distinguished by $\ominus$ ("Before"), $\mathcal{T}$ ("Trigger"), $\diamondminus$ ("Once"), and $\boxminus$ ("Historically"):

- $\ominus\varphi_1 \equiv \neg\ominus\neg\varphi_1$ [1]
- $\varphi_1\,\mathcal{T}\,\varphi_2 \equiv \neg(\neg\varphi_1\,\mathcal{S}\,\neg\varphi_2)$
- $\diamondminus\varphi_1 \equiv \mathit{true}\,\mathcal{S}\,\varphi_1$
- $\boxminus\varphi_1 \equiv \neg\diamondminus\neg\varphi_1 \equiv \mathit{false}\,\mathcal{T}\,\varphi_1$

[1] Since we are only observing infinite words, the $\bigcirc$-operator simply pushes $\neg$ further in, i.e., $\neg\bigcirc\varphi_1 \equiv \bigcirc\neg\varphi_1$.

Given some subset $X \subseteq AP$, the notation $\mathbb{B}^+(X)$ denotes the set of all *positive boolean formulas*, i.e., all formulas only build by conjunction and disjunction over the elements of X. Given a PLTL formula φ, the set of all *subformulas* of φ, including φ itself, is denoted by $Sub(\varphi)$. The set of all *principally temporal subformulas* of φ, i.e., with temporal connective as outermost operator, is denoted by $TSub(\varphi) \subset Sub(\varphi)$, possibly including φ itself. The *size* $|\varphi|$ of a formula φ is composed by the total number of its operators and atomic propositions.

2.2.2 Semantics

Let $i, j \in \mathbb{N}$ be two positions with $i \leq j$. The semantics of PLTL are defined over *infinite words* $\alpha = \alpha_0\alpha_1 \ldots \in (2^{AP})^\omega$ with letters $\alpha_i \in 2^{AP}$. Observing intervals, $\alpha[i, j]$ denotes the *finite substring* $\alpha_i \ldots \alpha_j$ of α. By $\alpha[i]$, we denote the *finite prefix* $\alpha[0, i-1] = \alpha_0\alpha_1 \ldots \alpha_{i-1}$, and, by α^i, we denote $\alpha_i\alpha_{i+1} \ldots$ as its *infinite suffix*. For a symbol α_i and a given set $X \in 2^{AP}$, $\alpha_i|_X$ denotes the *projection* of symbol α_i to a given set X, i.e., $\alpha_i|_X = \{p \in X \mid p \in \alpha_i\}$.

Given a PLTL formula φ and an infinite word $\alpha = \alpha_0\alpha_1 \ldots$, we define the relation $\alpha_i \vDash \varphi$, i.e., φ holds at position i of word α, recursively as follows:

- $\alpha_i \vDash p$ iff[2] $p \subseteq \alpha_i$ for $p \in 2^{AP}$
- $\alpha_i \vDash \neg\psi_1$ iff $\alpha_i \nvDash \psi_1$
- $\alpha_i \vDash \psi_1 \vee \psi_2$ iff $\alpha_i \vDash \psi_1$ or $\alpha_i \vDash \psi_2$
- $\alpha_i \vDash \psi_1 \wedge \psi_2$ iff $\alpha_i \vDash \psi_1$ and $\alpha_i \vDash \psi_2$
- $\alpha_i \vDash \bigcirc\psi_1$ iff $\alpha_{i+1} \vDash \psi_1$
- $\alpha_i \vDash \psi_1 \,\mathcal{U}\, \psi_2$ iff $\exists\, k \geq i.\, (\alpha_k \vDash \psi_2$ and $\forall\, i \leq j < k.\, \alpha_j \vDash \psi_1)$
- $\alpha_i \vDash \psi_1 \,\mathcal{R}\, \psi_2$ iff $\forall\, k \geq i.\, (\alpha_k \vDash \psi_2$ or $\exists\, i \leq j < k.\, \alpha_j \vDash \psi_1)$
- $\alpha_i \vDash \Diamond\psi_1$ iff $\exists\, k \geq i.\, \alpha_k \vDash \psi_1$
- $\alpha_i \vDash \Box\psi_1$ iff $\forall\, k \geq i.\, \alpha_k \vDash \psi_1$
- $\alpha_i \vDash \ominus\psi_1$ iff $i > 0$ and $\alpha_{i-1} \vDash \psi_1$
- $\alpha_i \vDash \circledcirc\psi_1$ iff $i = 0$ or $\alpha_{i-1} \vDash \psi_1$
- $\alpha_i \vDash \psi_1 \,\mathcal{S}\, \psi_2$ iff $\exists\, 0 \leq k \leq i.\, (\alpha_k \vDash \psi_2$ and $\forall\, k < j \leq i.\, \alpha_j \vDash \psi_1)$
- $\alpha_i \vDash \psi_1 \,\mathcal{T}\, \psi_2$ iff $\forall\, 0 \leq k \leq i.\, (\alpha_k \vDash \psi_2$ or $\exists\, k < j \leq i.\, \alpha_j \vDash \psi_1)$

[2] if, and only if

- $\alpha_i \vDash \diamondsuit\psi_1$ iff $\exists\, 0 \leq k \leq i.\, \alpha_k \vDash \psi_1$
- $\alpha_i \vDash \boxminus\psi_1$ iff $\forall\, 0 \leq k \leq i.\, \alpha_k \vDash \psi_1$

We call a word $\alpha \in (2^{AP})^\omega$ a *model* of a PLTL formula φ, denoted $\alpha \vDash \varphi$, if, and only if, $\alpha_0 \vDash \varphi$. Furthermore, a set of models M satisfies φ, denoted $M \vDash \varphi$, if, and only if, every model in M satisfies φ. Thus, a PLTL formula φ over AP defines a language over the alphabet 2^{AP}, given by

$$\mathcal{L}(\varphi) = \{\alpha \in (2^{AP})^\omega \mid \alpha \vDash \varphi\}.$$

Example. Below, we give three example formulas that take place in multiple parts of this thesis. ξ_1 is taken from the introduction on bounded synthesis [9], whereas ξ_2 is an extension of a widely used example for properties concerning the past [3, 18, 20, 21, 24]. Finally, ξ_3 gives a variation of the second specification, adding additional restrictions [7]. For all formulas, we assume a system that receives requests r_1, r_2, which it has to answer correctly by different types of grants g_1, g_2, respectively.

- ξ_1: Each request is eventually answered, and two grants are never given at the same time.

$$\xi_1 = \square(r_1 \rightarrow \diamondsuit g_1) \wedge \square(r_2 \rightarrow \diamondsuit g_2) \wedge \square\neg(g_1 \wedge g_2)$$

- ξ_2: Whenever a grant is encountered, it is preceded by its related request.

$$\xi_2 = \square(g_1 \rightarrow \diamondsuit r_1) \wedge \square(g_2 \rightarrow \diamondsuit r_2)$$

- ξ_3: For each request in the past, there is only a single grant (and a request is never answered at the same time step[3]).

$$\xi_3 = \square\Big(g_1 \rightarrow \big(\ominus(\neg g_1 \,\mathcal{S}\, r_1)\big)\Big)$$

2.2.3 Negative Normal Form

Let φ be a PLTL formula, its negative normal form (NNF) describes an equivalent formula φ' where negations are only applied on the level of atomic propositions. Furthermore, it only consists of atomic propositions (possibly negated), the boolean connectives $\vee, \wedge$, and the temporal connectives $\bigcirc, \ominus, \odot, \mathcal{U}, \mathcal{S}, \mathcal{R}, \mathcal{T}$.

[3] This restriction can be turned off by adding a disjunction, i.e., $\square\Big(g_1 \rightarrow \big(r_1 \vee \ominus(\neg g_1 \,\mathcal{S}\, r_1)\big)\Big)$. However, dropping this case makes it more interesting finding an implementation for our purposes.

Example. For our three example formulas, the NNFs would look like follows:

- $\xi'_1 = \bigwedge\limits_{i\in\{1,2\}} \Big(\mathit{false}\,\mathcal{R}\,\big(\neg r_i \vee (\mathit{true}\,\mathcal{U}\,g_i)\big)\Big) \wedge \mathit{false}\,\mathcal{R}\,(\neg g_1 \vee \neg g_2)$
- $\xi'_2 = \Big(\mathit{false}\,\mathcal{R}\,\big(\neg g_1 \vee (\mathit{true}\,\mathcal{S}\,r_1)\big)\Big) \wedge \Big(\mathit{false}\,\mathcal{R}\,\big(\neg g_2 \vee (\mathit{true}\,\mathcal{S}\,r_2)\big)\Big)$
- $\xi'_3 = \mathit{false}\,\mathcal{R}\,\Big(\neg g_1 \vee \big(\ominus(\neg g_1\,\mathcal{S}\,r_1)\big)\Big)$

2.2.4 Laws and Equivalences Rules

For the following rules, let $\varphi_1, \ldots, \varphi_4$ be arbitrary PLTL formulas. Based on the definition of the temporal binary operators $\mathcal{U}$ and $\mathcal{S}$, the below defined *expansion laws* describe the unfolding of a single time step:

$$\begin{aligned} \varphi_1\,\mathcal{U}\,\varphi_2 &\equiv \varphi_2 \vee \big(\varphi_1 \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)\big) \\ \varphi_1\,\mathcal{S}\,\varphi_2 &\equiv \varphi_2 \vee \big(\varphi_1 \wedge \ominus(\varphi_1\,\mathcal{S}\,\varphi_2)\big) \end{aligned} \tag{2.1}$$

The definitions of the $\mathcal{R}$- and the $\mathcal{T}$-operator, respectively, allow an alternative formulation:

$$\begin{aligned} \varphi_1\,\mathcal{R}\,\varphi_2 &\equiv \big(\Box\varphi_2\big) \vee \big(\varphi_2\,\mathcal{U}\,(\varphi_1 \wedge \varphi_2)\big) \\ \varphi_1\,\mathcal{T}\,\varphi_2 &\equiv \big(\boxminus\varphi_2\big) \vee \big(\varphi_2\,\mathcal{S}\,(\varphi_1 \wedge \varphi_2)\big) \end{aligned} \tag{2.2}$$

There are multiple distributivity laws between conjunction, disjunction, and the temporal operators $\mathcal{U}$ and $\mathcal{S}$. Both temporal connectives are right-distributive with disjunction and left-distributive with conjunction:

$$\begin{aligned} \varphi_1\,\mathcal{U}\,(\varphi_2 \vee \varphi_3) &\equiv (\varphi_1\,\mathcal{U}\,\varphi_2) \vee (\varphi_1\,\mathcal{U}\,\varphi_3) \\ \varphi_1\,\mathcal{S}\,(\varphi_2 \vee \varphi_3) &\equiv (\varphi_1\,\mathcal{S}\,\varphi_2) \vee (\varphi_1\,\mathcal{S}\,\varphi_3) \\ (\varphi_1 \wedge \varphi_2)\,\mathcal{U}\,\varphi_3 &\equiv (\varphi_1\,\mathcal{U}\,\varphi_3) \wedge (\varphi_2\,\mathcal{U}\,\varphi_3) \\ (\varphi_1 \wedge \varphi_2)\,\mathcal{S}\,\varphi_3 &\equiv (\varphi_1\,\mathcal{S}\,\varphi_3) \wedge (\varphi_2\,\mathcal{S}\,\varphi_3) \end{aligned} \tag{2.3}$$

Furthermore, two $\mathcal{U}$-statements with a conjunction in their top-level are equivalent to a disjunction in their top-level and conjunctions in their sub-level. The same holds for the $\mathcal{S}$-operator:

$$\begin{aligned} (\varphi_1\,\mathcal{U}\,\varphi_2) \wedge (\varphi_3\,\mathcal{U}\,\varphi_4) \equiv &\Big((\varphi_1 \wedge \varphi_3)\,\mathcal{U}\,(\varphi_2 \wedge \varphi_4)\Big) \\ &\vee \Big((\varphi_1 \wedge \varphi_3)\,\mathcal{U}\,\big(\varphi_2 \wedge (\varphi_3\,\mathcal{U}\,\varphi_4)\big)\Big) \\ &\vee \Big((\varphi_1 \wedge \varphi_3)\,\mathcal{U}\,\big(\varphi_4 \wedge (\varphi_1\,\mathcal{U}\,\varphi_2)\big)\Big) \end{aligned} \tag{2.4}$$

2.2.5 PLTL vs. LTL, and Gabbay's Separation Theorem

As already mentioned, PLTL is proven to be more succinct than standard LTL. In [17, 21], a family of PLTL formulas $\{\varphi_n\}_{n\in\mathbb{N}}$ with size $O(n)$ is presented, whose expression in LTL without an exponential blowup is infeasible. Formally, for atomic propositions $p_0, \ldots, p_n$ with $n \in \mathbb{N}$, the formula

$$\varphi_n = \Box\Big(\big(\bigwedge_{i=1}^{n}(p_i \leftrightarrow \diamondsuit\boxminus p_i)\big) \rightarrow (p_0 \leftrightarrow \diamondsuit\boxminus p_0)\Big)$$

describes that any position of a path that agrees on the propositions $p_1, \ldots, p_n$ with the initial state, also agrees on p_0. The proof is done by contradiction using the assumption that there is an LTL formula φ_n' that is equivalent to φ_n using polynomial size. Disproving this statement results in an LTL formula φ_n' of size at least $O(2^n)$ and, thus, PLTL is more succinct than without its past connectives.

However, as it is shown exemplarily by the theorem above, PLTL is as expressive as LTL [10] and, therefore, for each PLTL formula φ there is an equivalent LTL formula φ' of *non-elementary* complexity. To be concrete, there is a clear separation $\varphi'' \in \mathbb{B}^+(\varphi_{\text{future}}, \varphi_{\text{present}}, \varphi_{\text{past}})$ between future, present, and past, so that using the fact that there is no past in the initial time step, deleting φ_{past} results in an initial equivalent formula $\varphi \equiv \varphi'' \equiv_i \varphi' \in \mathbb{B}^+(\varphi_{\text{future}}, \varphi_{\text{present}})$. Two formulas φ and φ' are said to be *initial equivalent* ($\varphi \equiv_i \varphi'$) if they agree on all words in $\mathcal{L}(\varphi)$ and $\mathcal{L}(\varphi')$, i.e., $\varphi \equiv_i \varphi'$, if, and only if, for all words $\alpha \in (2^{AP})^\omega$, $\alpha_0 \vDash \varphi$ iff $\alpha_0 \vDash \varphi'$.

The statement is proven in [10] by Gabbay, known as *Gabbay's Separation Theorem*. Gabbay presents eight elimination rules pushing $\mathcal{U}$ out of the scope of $\mathcal{S}$, which can be directly mirrored by time-equivalence of future and past operators. Afterwards, he proves completeness of these cases by using a proof over the structure of the formula.

A drawback of Gabbay's Theorem in [10] is the usage of strict temporal operators for its formulation. Strict PLTL is equivalent to standard PLTL, but it reduces the number of temporal operators to two, making it possible to express every PLTL formula by boolean connectives and the strict versions of $\mathcal{U}$ and $\mathcal{S}$, abbreviated by $\widehat{\mathcal{U}}$ ("Strict Until") and $\widehat{\mathcal{S}}$ ("Strict Since"). The idea is that a principally temporal formula using strict PLTL is satisfied in a position if the previous or succeeding position satisfies the non-strict version. Formally, for some position $i \in \mathbb{N}$ of word α, and arbitrary strict PLTL formulas ψ_1 and ψ_2:

- $\alpha_i \vDash \psi_1 \,\widehat{\mathcal{U}}\, \psi_2$ iff $\exists\, k > i.\, (\alpha_k \vDash \psi_2$ and $\forall\, i < j < k.\, \alpha_j \vDash \psi_1)$
- $\alpha_i \vDash \psi_1 \,\widehat{\mathcal{S}}\, \psi_2$ iff $\exists\, 0 \leq k < i.\, (\alpha_k \vDash \psi_2$ and $\forall\, k < j < i.\, \alpha_j \vDash \psi_1)$

The operators $\bigcirc$ and $\ominus$ can be abbreviated by $\mathit{false}\ \widehat{\mathcal{U}}\ \psi$ and $\mathit{false}\ \widehat{\mathcal{S}}\ \psi$, respectively.

In Chapter 3, we present a rewriting of Gabbay's Separation Theorem using standard PLTL.

Example. We show the initially equivalent LTL formulas for the PLTL example formulas ξ_2 and ξ_3.

- $\xi_2 \equiv_i \neg\big(\neg r_1\,\mathcal{U}\,(g_1 \wedge \neg r_1)\big) \wedge \neg\big(\neg r_2\,\mathcal{U}\,(g_2 \wedge \neg r_2)\big)$
- $\xi_3 \equiv_i \big(r_1\,\mathcal{R}\neg g_1\big) \wedge \Box\big(g_1 \rightarrow \bigcirc(r_1\,\mathcal{R}\neg g_1)\big)$

2.3 Implementations

In this section, we present the depiction of implementations by using *labeled transition systems* (TSs), extended by a set of directions, called *directed transition systems* (DTSs). We refine the representation of TSs in the context of *fairness-free discrete systems* (FFDSs), which finally lead to the definition of *temporal testers* (TTs). The definitions are based on [8] and adapted by using [9] for the definition of DTSs, as well as [4] and [15] for the depiction of FFDSs and TTs. For simplification, if not stated otherwise, we assume all sequences to be infinite.

2.3.1 Directed Transition Systems

A Σ-labeled Υ-transition system (DTS) $\mathcal{T}$ is a four-tuple, where states are labeled by elements of a finite alphabet Σ, and transitions are directed over a finite set of directions Υ. For given sets Σ, Υ, it is defined by

$$\mathcal{T} = (T, t_0, \tau, o)$$

where

- T is a set of states,
- $t_0 \in T$ is the initial state,
- $\tau : T \times \Upsilon \rightarrow T$ is the transition relation, and
- $o : T \rightarrow \Sigma$ is the labeling function.

We call $\mathcal{T}$ *finite-state transition system* if, and only if, T is finite. An *execution* of a Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is an infinite sequence of states $\pi = t_0 t_1 \ldots \in T^\omega$ such that π is initial, i.e., starting in initial state t_0, and satisfies the transition relation, i.e., $\forall\, i \in \mathbb{N}.\, \exists\, \upsilon \in \Upsilon.\, \tau(t_i, \upsilon) = t_{i+1}$.

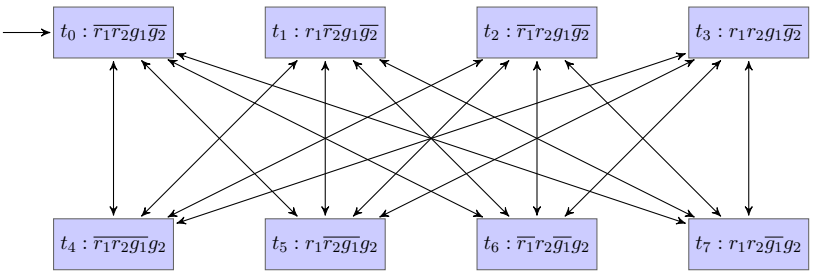


Figure 2.2: Implementation $\mathcal{T}_{\xi_1}$ for ξ_1 as input-preserving $2^{\{r_1,r_2,g_1,g_2\}}$-labeled $2^{\{r_1,r_2\}}$-transition system with no request as initial input. Each state t is labeled by its labeling function $o(t)$. Its strategy alternates between giving grants g_1 and g_2. Thus, each request will be answered eventually without a simultaneous granting.

Ongoing, DTSs describe implementations of reactive systems for a given set AP of atomic propositions, which is partitioned into a set I of boolean inputs, describing its directions, and a set O of boolean outputs, such that $AP = I \dot{\cup} O$. Thus, we are interested in 2^{AP}-labeled 2^I-transition systems with some fixed initial input $i_0 \subseteq I$.

A 2^{AP}-labeled 2^I-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is called *input-preserving* if the label of each state accurately reflects the previous input, i.e., the initial input is reflected by the initial state, $o(t_0) \cap I = i_0$, and for all states $t \in T$ and inputs $i \subseteq I$, we have $o(\tau(t, i)) \cap I = i$. Furthermore, $\mathcal{T}$ *satisfies* a PLTL formula φ if, for each execution $\pi = t_0 t_1 \ldots \in T^\omega$ over $\mathcal{T}$, the sequence $\sigma_\pi = o(t_0)o(t_1) \ldots \in (2^{AP})^\omega$ is a model of φ, denoted $\sigma_\pi \vDash \varphi$. We call sequence σ_π the *trace* of execution π.

Example. Figure 2.2 shows an implementation for ξ_1, Figure 2.3 and Figure 2.4 implementations for ξ_2 and ξ_3, respectively. In the following, the implementations are indicated by their related formulas, i.e., $\mathcal{T}_\varphi$ with $\varphi \in \{\xi_1, \xi_2, \xi_3\}$. All DTSs are input-preserving since each state labeling reflects its previous input.

2.3.2 Fairness-Free Discrete Systems (FFDSs)

A *fairness-free discrete system* (FFDS) builds a higher-level symbolic representation of a TS implementing pure-past PLTL formulas. States describe interpretations of variables of the system that range over discrete domains, and transitions describe the changes of these. For simplicity, we can assume without loss of generality (w.l.o.g.) that all variables of the system range over the boolean domain. Thus, an FFDS is a three-tuple

$$\mathcal{D} = (V, \theta, \rho)$$

where

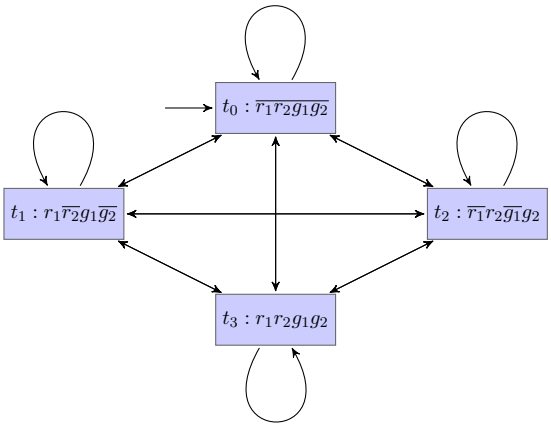


Figure 2.3: Implementation T_{ξ_2} for ξ_2 as input-preserving $2^{\{r_1,r_2,g_1,g_2\}}$-labeled $2^{\{r_1,r_2\}}$-transition system with no request as initial input. Each state t is labeled by its labeling function $o(t)$. Its strategy is that whenever there is a request, it is immediately answered by its related grant.

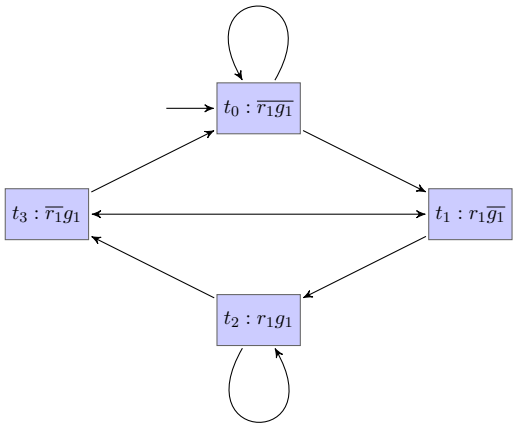


Figure 2.4: Implementation T_{ξ_3} for ξ_3 as input-preserving $2^{\{r_1,g_1\}}$-labeled $2^{\{r_1\}}$-transition system with no request as initial input. Its strategy ensures that a grant is not given in the same time step as the request is. Therefore, it is granted in the succeeding time step, additionally taking care about another request.

- $V = \{v_1, \ldots, v_n\}$ is an *alphabet* as finite set of boolean variables. The set of states Σ_V is defined as the *set of interpretations* of V, i.e., $\Sigma_V = 2^V$.

- θ is the *initial condition*, which is an assertion over V that characterizes all the initial states of $\mathcal{D}$. A state $s \in \Sigma_V$ is called *initial* if it satisfied θ, denoted $s \vDash \theta$.

- ρ is the *transition relation*. It describes an assertion over $V \cup V'$, where V' is a primed copy of the variables in V, describing the successor states. Thus, ρ relates a state $s \in \Sigma_V$ to its $\mathcal{D}$-successors $s' \in \Sigma_{V'}$, leading to $(s, s') \vDash \rho$.

A *computation* π of an FFDS $\mathcal{D}$ is an infinite sequence of states $s_0 s_1 \ldots \in (2^V)^\omega$ such that $s_0 \vDash \theta$ and $\forall\, i \in \mathbb{N}.\, (s_i, s_{i+1}) \vDash \rho$. Furthermore, FFDS $\mathcal{D}$ is said to *implement* a PLTL specification φ, denoted $\mathcal{D} \vDash \varphi$, if each computation of $\mathcal{D}$ satisfies φ.

As for DTS, we can partition the alphabet of FFDS $\mathcal{D} = (V, \theta, \rho)$ into a set I of input and a set O of output variables, i.e., $V = I \dot\cup O$. $\mathcal{D}$ is called *deterministic* with respect to its input set $I \subseteq V$, if

- $\mathcal{D}$ has deterministic initial states, i.e., $\forall\, s, t \in \Sigma_V.$ if $s \vDash \theta$ and $t \vDash \theta$ and $s|_I = t|_I$, then $s = t$, and

- $\mathcal{D}$ has deterministic transitions, i.e., $\forall\, s, s', s'' \in \Sigma_V.$ if $(s, s') \vDash \rho, (s, s'') \vDash \rho$, and $s'|_I = s''|_I$, then $s' = s''$ holds.

$\mathcal{D}$ is called *complete* with respect to its input set $I \subseteq V$, if

- $\mathcal{D}$'s initial states reflect all inputs, i.e., $\forall\, s_i \in \Sigma_I.\, \exists\, s \in \Sigma_V.\, s|_I = s_i$ and $s \vDash \theta$, and

- $\mathcal{D}$'s transitions reflect all inputs, i.e., $\forall\, s \in \Sigma_V, s'_i \in \Sigma_I.\, \exists\, s' \in \Sigma_V.\, s'|_I = s'_i$ and $(s, s') \vDash \rho$.

Given FFDS $\mathcal{D}$ that is deterministic and complete with respect to input set I, for every possible sequence of inputs $\sigma = i_0 i_1 \ldots \in (\Sigma_I)^\omega$, $\mathcal{D}$ has a unique computation $\pi = s_0 s_1 \ldots$ such that $\forall\, j \geq 0.\, i_j = s_j|_I$ holds. We call π the computation of $\mathcal{D}$ on σ.

2.3.3 Synchronous Parallel Composition of FFDSs

It is possible to combine two given FFDSs in order to create a single FFDS of greater size. This can be done either in a *synchronous* or an *asynchronous* way. Whereas the first considers joint transitions of the two systems at once, the second allows to apply changes just in one of the systems. For this thesis,

we are interested in the synchronous parallel composition. Given two FFDSs $\mathcal{D}_1 = (V_1, \theta_1, \rho_1)$ and $\mathcal{D}_2 = (V_2, \theta_2, \rho_2)$, the *synchronous parallel composition* of $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted $\mathcal{D}_1 \,|||\, \mathcal{D}_2$, results in a new FFDS defined by

$$\mathcal{D}_1 \,|||\, \mathcal{D}_2 = (V_1 \cup V_2, \theta_1 \wedge \theta_2, \rho_1 \wedge \rho_2).$$

Its idea is that both systems share inputs and outputs and react synchronous. However, their initial and transition assertions have to hold at once in order to step further.

2.3.4 Temporal Testers as FFDSs

Given a pure-past PLTL formula φ, one can construct an FFDS $\mathcal{D}_\varphi$, called a *temporal tester* (TT) for φ, that monitors the truth values of φ in the states of a computation over a set of atomic propositions AP. For this, AP additionally contains a set of distinguished boolean variables for each temporal subformula of φ, as well as variable $x_\varphi \in \mathbb{B}$ for φ itself, such that

- $\mathcal{D}_\varphi$ is complete with respect to AP_φ,
- For every computation $\pi = t_0 t_1 \ldots \in (2^{AP})^\omega$ of $\mathcal{D}_\varphi$, we have

  $$t_i[x_\varphi] = \mathit{true} \text{ iff } (\pi, i) \vDash \varphi,$$

  where $t_i[x_\varphi]$ denotes the truth value of x_φ in position i of π, and
- For every sequence of states $\sigma = s_0 s_1 \ldots \in (2^{AP_\varphi})^\omega$, there is a computation $\pi = t_0 t_1 \ldots \in (2^{AP})^\omega$ of $\mathcal{D}_\varphi$, such that for every $i \geq 0$, we have

  $$t_i|_{AP_\varphi} = s_i,$$

  i.e., σ builds a projection of the states of π, which contain the additional boolean variables, onto the alphabet of φ.

Construction 1. Given a pure-past PLTL formula φ, the temporal tester $\mathcal{D}_\varphi$ for φ is denoted as an FFDS $\mathcal{D}_\varphi = (V_\varphi, \theta_\varphi, \rho_\varphi)$, where its components are inductively defined over the structure of the formula φ as follows:

- The *system variables* V_φ are build by the union of AP_φ and the set of additional variables for each principally temporal subformula $\psi \in TSub(\varphi)$. Additionally, if φ is not principally temporal, we add another variable x_φ. Thus:

  $$V_\varphi = AP_\varphi \cup \left\{ x_\psi \mid \psi \in TSub(\varphi) \cup \{\varphi\} \right\}$$

  In the following, $X \subset V_\varphi$ denotes the set of all x-variables in V_φ.

For the other components, we define an inductive formula χ that maps each subformula $\psi \in Sub(\varphi)$ of φ to an assertion over V_φ:

$$\chi(\psi) = \begin{cases} p, & \text{if } \psi = p \in AP_\varphi \\ \neg\chi(\psi_1), & \text{if } \psi = \neg\psi_1 \\ \chi(\psi_1) \vee \chi(\psi_2), & \text{if } \psi = \psi_1 \vee \psi_2 \\ \chi(\psi_1) \wedge \chi(\psi_2), & \text{if } \psi = \psi_1 \wedge \psi_2 \\ x_\psi, & \text{if } \psi \in TSub(\varphi) \end{cases}$$

- For the *initial condition* θ_φ, we have to make additional observations on the principally temporal subformulas of φ concerning the past. Each formula regarding the previous state is initially $false$, since there is no past yet. Each $\mathcal{S}$-subformula is mapped to $true$ if its rightside holds in the initial state. If formula φ is not principally temporal, its variable is initially determined using function χ. This leads to the assertion θ_φ with:

$$\theta_\varphi := \big(x_\varphi \leftrightarrow \chi(\varphi)\big) \wedge \bigwedge_{\ominus\psi \in Sub(\varphi)} \neg x_{\ominus\psi} \wedge \bigwedge_{\psi_1 \mathcal{S} \psi_2 \in Sub(\varphi)} \big(x_{\psi_1 \mathcal{S} \psi_2} \leftrightarrow \chi(\psi_2)\big)$$

- The *transition relation* ρ_φ has to make sure that the variables x_ψ for each principally temporal subformula ψ are updated correctly. Therefore, the temporal unary operator $\ominus$ observes the previous state of the computation. The temporal binary operator $\mathcal{S}$ follows the idea of the expansion laws (see Equation 2.1). If formula φ is not principally temporal, one has to make sure that its boolean operator at top-level is considered as well, using previously defined function χ. Formally, assertion ρ_φ is defined by:

$$\rho_\varphi := \begin{cases} & \big(x'_\psi \leftrightarrow \chi'(\psi)\big) \\ \wedge & \bigwedge_{\ominus\psi \in Sub(\varphi)} \big(x'_{\ominus\psi} \leftrightarrow \chi(\psi)\big) \\ \wedge & \bigwedge_{\psi_1 \mathcal{S} \psi_2 \in Sub(\varphi)} \Big(x'_{\psi_1 \mathcal{S} \psi_2} \leftrightarrow \chi'(\psi_2) \vee \big(\chi'(\psi_1) \wedge x_{\psi_1 \mathcal{S} \psi_2}\big)\Big) \end{cases}$$

Example. Since ξ_2 and ξ_3 are formulas of the *future(past)-fragment* of PLTL, i.e., a pure-past formula is only surrounded by future-temporal and boolean connectives, we give the temporal testers for their inner past formulas.

For $\xi_2^1 = \diamondminus r_1 = true\, \mathcal{S}\, r_1$ and $\xi_2^2 = \diamondminus r_2 = true\, \mathcal{S}\, r_2$, the components of $\mathcal{D}_{\xi_2^i} = (V_{\xi_2^i}, \theta_{\xi_2^i}, \rho_{\xi_2^i})$ with $i \in \{1, 2\}$ are defined as follows:

- $V_{\xi_2^i} = \{r_i\} \cup \{x_{\Diamond r_i}\}$
- $\theta_{\xi_2^i} := x_{\Diamond r_i} \leftrightarrow r_i$
- $\rho_{\xi_2^i} := x'_{\Diamond r_i} \leftrightarrow r'_i \vee x_{\Diamond r_i}$

By using the synchronous parallel composition $\mathcal{D}_{\xi_2^1} \mid\mid\mid \mathcal{D}_{\xi_2^2}$, presented in Section 2.3.3, we obtain temporal tester $\mathcal{D}_{\xi'_2} = (V_{\xi'_2}, \theta_{\xi'_2}, \rho_{\xi'_2})$ with:

- $V_{\xi'_2} = \{r_1, r_2\} \cup \{x_{\Diamond r_1}, x_{\Diamond r_2}\}$
- $\theta_{\xi'_2} := (x_{\Diamond r_1} \leftrightarrow r_1) \wedge (x_{\Diamond r_2} \leftrightarrow r_2)$
- $\rho_{\xi'_2} := (x'_{\Diamond r_1} \leftrightarrow r'_1 \vee x_{\Diamond r_1}) \wedge (x'_{\Diamond r_2} \leftrightarrow r'_2 \vee x_{\Diamond r_2})$

For $\xi_3^1 = \ominus(\neg g_1 \,\mathcal{S}\, r_1)$, we define $\mathcal{D}_{\xi'_3} = (V_{\xi'_3}, \theta_{\xi'_3}, \rho_{\xi'_3})$ by the following components:

- $V_{\xi'_2} = \{r_1, g_1\} \cup \{x_{\neg g_1 \mathcal{S} r_1}, x_{\ominus(\neg g_1 \mathcal{S} r_1)}\}$
- $\theta_{\xi'_2} := \neg x_{\ominus(\neg g_1 \mathcal{S} r_1)} \wedge (x_{\neg g_1 \mathcal{S} r_1} \leftrightarrow r_1)$
- $\rho_{\xi'_2} := \left(x'_{\neg g_1 \mathcal{S} r_1} \leftrightarrow r'_1 \vee (g'_1 \wedge x_{\neg g_1 \mathcal{S} r_1})\right) \wedge \left(x'_{\ominus(\neg g_1 \mathcal{S} r_1)} \leftrightarrow x_{\neg g_1 \mathcal{S} r_1}\right)$

2.4 Automata Theory

To describe languages over infinite words automata are introduced. Their components are very similar to those of automata over finite words. However, their acceptance is defined on infinite behavior. We distinguish automata over words and trees. Word automata describe a single sequence over a given alphabet that is accepted if it is part of the acceptance condition. Tree automata are word automata with additional directions on each state such that a sequence defines a tree. Thus, it is possible to describe a word automaton as a tree automaton with a single direction. After defining their components in detail, we list the considered acceptance conditions before presenting special properties over automata. The definition of word automata, properties, and acceptance conditions are taken from [13]. For introducing tree automata, we use the definitions presented in [5].

2.4.1 Infinite Word Automata

An *automaton* over *infinite words* is defined by the five-tuple

$$\mathcal{A} = (\Sigma, Q, Q_0, \delta, Acc)$$

where

- Σ is a finite alphabet,

- Q is a finite set of states,
- $Q_0 \in Q$ is a set of initial states,
- $\delta : Q \times \Sigma \to Q$ is the transition function, and
- $Acc \subseteq Q^\omega$ describes the set of accepted sequences, called accepting condition.

A *run* r on a word automaton $\mathcal{A}$ over an infinite word $\alpha \in \Sigma^\omega$ is an infinite sequence of states $q_0 q_1 \ldots$ such that it is starting with an initial state $q_0 \in Q_0$, and fulfills the transition function, i.e., $\forall\, i \in \mathbb{N}.\, \delta(q_i, \alpha_i) = q_{i+1}$. We say that a run r is *accepting* if $r \in Acc$. The *set of all occurring states* in a run r, denoted by $Occ(r)$, is defined by

$$Occ(r) = \{q \in Q \mid \exists\, n \in \mathbb{N}.\, q_n = q\}.$$

$Inf(r)$ describes *all infinite occurring states* in a run $r = q_0 q_1 \ldots$, formally

$$Inf(r) = \{q \in Q \mid \forall\, m \in \mathbb{N}.\, \exists\, n \in \mathbb{N}.\, n > m \text{ and } q_m = q = q_n\}$$

Similar, we define the *set of all infinite occurring letters* in a word $\alpha \in \Sigma^\omega$ by

$$Inf(\alpha) = \left\{\sigma \in \Sigma \mid \forall\, m \in \mathbb{N}.\, \exists\, n \in \mathbb{N}.\, n > m \text{ and } \alpha_m = \sigma = \alpha_n\right\}$$

Finally, the *language* of a word automaton $\mathcal{A}$ is defined by

$$\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}.$$

2.4.2 Infinite Trees and Infinite Tree Automata

To define infinite tree automata, we first introduce their inputs as trees. An X-labeled Υ-directed *tree* over a set of symbols X and a set of directions Υ is a function $\gamma : \Upsilon^\infty \to X$ where each (possibly finite) branch $x \in \Upsilon^\infty$, called a *node* in the tree, is assigned a labeling $\gamma(x) \in X$. Note that, if not stated otherwise, we assume all branches of a tree to be extended to infinity, called *infinite tree*. The *root* in a tree is build by the empty branch ϵ. A tree is *completely connected*, i.e., if a node $x \in \Upsilon^*$ is not labeled by γ, so are none of its successors, formally:

$$\forall\, x \in \Upsilon^*.\, \forall\, x' \in \Upsilon^\infty.\, \gamma(x) = \text{undef.} \to \gamma(x \cdot x') = \text{undef.}$$

An *automaton* over *infinite directed trees* is defined as an automaton over infinite words, extended by a set of directions to each state. Thus,

$$\mathcal{A} = (\Sigma, \Upsilon, Q, Q_0, \delta, Acc),$$

where

- Σ is a finite alphabet,
- Υ is a finite set of directions,
- Q is a finite set of states,
- $Q_0 \in Q$ is a set of initial states,
- $\delta : Q \times \Sigma \to 2^{Q \times \Upsilon}$ is the transition function, where it assigns for each direction $\upsilon \in \Upsilon$ at most a single state $q \in Q$, building the tuple $(q, \upsilon) \in Q \times \Upsilon$, formally:

$$\forall\, q \in Q. \forall\, \sigma \in \Sigma. \forall\, \upsilon \in \Upsilon. \, |(\cdot, \upsilon) \in \delta(q, \sigma)| \leq 1,$$

  and

- $Acc \subseteq Q^\omega$ is the accepting condition.

A *run* of a tree automaton $\mathcal{A}$ on a Σ-labeled Υ-directed infinite tree $\gamma_\Sigma : \Upsilon^\omega \to \Sigma$ is an Υ-directed Q-labeled infinite tree $\gamma_Q : \Upsilon^\omega \to Q$ such that $\gamma_Q(\epsilon) \in Q_0$ and $\forall\, x \in \Upsilon^*. \forall\, \upsilon \in \Upsilon. \big(\gamma_Q(x \cdot \upsilon), \upsilon\big) \in \delta\big(\gamma_Q(x), \gamma_\Sigma(x)\big)$. We call such a run γ_Q *accepting* if every infinite branch $x = x_0 x_1 \ldots$ satisfies the accepting condition, i.e., $\gamma_Q(x_0)\gamma_Q(x_0 x_1) \ldots \in Acc$.

Note: If not stated otherwise, we assume for all automata that their set of initial states Q_0 is a singleton. This assumption is possible since each automaton with multiple initial states has an equivalent automaton with a single distinguishable initial state that simulates all the previous ones. The construction increases the overall number of states by 1.

2.4.3 Acceptance Conditions

For infinite word and tree automata with state space Q, we define a couple of accepting conditions:

The *Safety acceptance condition* Safety(S) is given by a set of safe states $S \subseteq Q$. A run of an infinite word automaton or a branch of an infinite tree automaton is called *safe* if it stays in S. Formally,

$$\text{Safety}(S) = \{\alpha \in Q^\omega \mid Occ(\alpha) = S\}.$$

The respective Safety automata are called *Safety word automaton* (SW) and *Safety tree automaton* (ST).

The *Büchi acceptance condition* Büchi(F) is given by a set of accepting states $F \subseteq Q$. A run of an infinite word automaton or a branch of an

infinite tree automaton is *Büchi accepting* if some state from its set F occurs infinitely often. Formally,

$$\text{Büchi}(F) = \{\alpha \in Q^\omega \mid Inf(\alpha) \cap F \neq \emptyset\}.$$

The respective Büchi automata are called *Büchi word automaton* (BW) and *Büchi tree automaton* (BT).

The dual of the Büchi acceptance condition is the *co-Büchi acceptance condition* coBüchi(C), $C \subseteq Q$. In here, a run or a branch is *co-Büchi accepting* if the set C is only visited finitely often. Formally,

$$\text{coBüchi}(C) = \{\alpha \in Q^\omega \mid Inf(\alpha) \cap C = \emptyset\}.$$

We talk about a *co-Büchi word automaton* (CW) and a *co-Büchi tree automaton* (CT).

Since Büchi and co-Büchi are dual, we are able to transform a Büchi automaton into coBüchi, and vice versa. To do so, it suffices to build the counterset of the given set and change the condition, i.e., for given sets $F, C \subseteq Q$, Büchi(F) = coBüchi($Q \setminus F$), as well coBüchi(C) = Büchi($Q \setminus C$).

The *Generalized Büchi acceptance condition* GenBüchi($\mathcal{F}$) is given by a set of sets of states $\mathcal{F} \subseteq 2^Q$. A run or a branch is called *Generalized Büchi accepting* if for each set $F \in \mathcal{F}$ some state from F occurs infinitely often. Formally,

$$\text{GenBüchi}(\mathcal{F}) = \{\alpha \in Q^\omega \mid \forall\, F \in \mathcal{F}.\, Inf(\alpha) \cap F \neq \emptyset\}.$$

The respective automata are called *Generalized Büchi word automaton* (GBW) and *Generalized Büchi tree automaton* (GBT).

For Büchi, co-Büchi, and Generalized Büchi, it is possible to define the acceptance conditions over a set or a set of sets of transitions, called *transition-based*. Rather than observing sequences of states, one has to consider sequences of transitions and observe the set of infinitely occurring transitions.

Finally, the *Parity acceptance condition* Parity(c) is given by a so called *coloring function* $c : Q \to \mathcal{C}$, where c is projecting onto a set of colors $\mathcal{C} \subseteq \mathbb{N}$. A run or a branch $r = q_0 q_1 \ldots$ is called *Parity accepting* if the highest number occurring infinitely often in the sequence $c(q_0)c(q_1)\ldots \in \mathcal{C}^\omega$ is even. Formally,

$$\text{Parity}(c) = \big\{\alpha \in Q^\omega \mid \max\{c(q) \mid q \in \text{Inf}(\alpha)\} \text{ is even}\big\}.$$

The respective automata are called *Parity word automaton* (PW) and *Parity tree automaton* (PT).

2.4.4 Automata Properties

Instead of considering the outcome of transition relations as an element of a set, one could describe it as an assertion over this set. Thus, an infinite word automaton $\mathcal{A} = (\Sigma, Q, Q_0, \delta, Acc)$ is called *alternating* if its transition relation is of the form

$$\delta : Q \times \Sigma \to \mathbb{B}^+(Q).$$

Similary, an infinite tree automaton $\mathcal{A} = (\Sigma, \Upsilon, Q, Q_0, \delta, Acc)$ is called *alternating* if its transition relation is of the form

$$\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon).$$

A *universal* automaton describes a special kind of alternating automaton where the mentioned assertion is only build by conjunctions. Accordingly, in a *non-deterministic* automaton the assertions are build only using disjunctions. Finally, a universal and non-determinstic automaton is called *deterministic*, i.e., the outcome of the transition relation is a singleton.

For abbreviations, we assigned "A" as a prefix to alternating automata, "U" to universal automata, "N" to non-deterministic automata, and "D" to deterministic automata, respectively. Thus, e.g., for a co-Büchi tree automaton, we would distinguish the abbreviations ACT, UCT, NCT, DCT.

Example. ξ_1 as a universal co-Büchi tree automaton (UCT) is given by

$$\mathcal{U}_{\xi_1} = \big(\{r_1, r_2, g_1, g_2\}, \{r_1, r_2\}, \{q_0, q_1, q_2, \bot\}, \{q_0\}, \delta, \text{coBüchi}(\{q_1, q_2\})\big).$$

Its transitions can be observed in Figure 2.5.

2.5 Tableau Construction

In [12], Gastin and Oddoux present a tableau construction that creates an NBW starting from a PLTL formula. Given some PLTL formula φ with $|\varphi| = n$, they create a *progressing two-way very-weak alternating Büchi automaton* (2VWABA) of size $n + 1$. Afterwards, they translate it to a GBW and, using a *round-robin construction*, finally to an NBW, leading to an exponential blowup in total.

2.5.1 Two-Way Very-Weak Alternating Aut. (2VWAA)

A special variation of tree automata is called *two-way automata* (2A), where the set of directions Υ consists of just two elements 1 and -1, indicating whether the head moves forward or backward on a given input word (instead

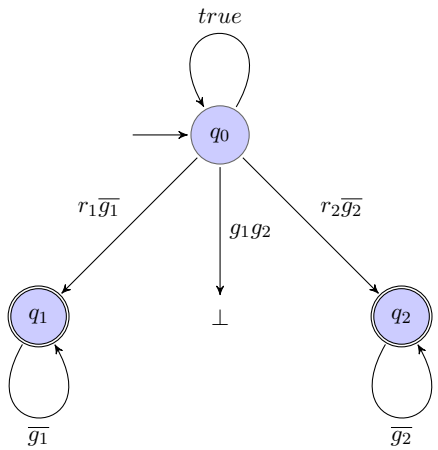


Figure 2.5: UCT $\mathcal{U}_{\xi_1}$ for ξ_1. Double-circles indicate the rejecting states of the co-Büchi condition. The transition labeling *true* indicates the set of all labels. As soon as it encounters a request that is not answered by the related grant, it stays in the rejecting state until it is granted. Two simultaneous grants would lead to a dead-end, meaning that there is a finite branch and, thus, a run would not be accepting any more.

of an input tree), respectively. In here, we directly observe two-way alternating automata. A *two-way alternating automaton* (2AA) $\mathcal{A}$ with $\Upsilon = \{-1, 1\}$ is a six-tuple

$$\mathcal{A} = (\Sigma, Q, Q_0, \delta, Fin, Acc)$$

where

- Σ is a finite alphabet,
- Q is a finite set of states,
- $Q_0 \in Q$ is a set of initial states,
- $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \{-1, 1\})$ is the transition function,
- $Fin \subseteq Q$ is the set of final states, and
- $Acc \subseteq Q^\omega$ is the accepting condition.

Since its implicit set of directions Υ only consists of two elements, there is an alternative representation for its transition function by

$$\delta : Q \to 2^{2^Q \times 2^\Sigma \times 2^Q}.$$

It says that for a resulting tuple $(\overleftarrow{X}, \sigma, \overrightarrow{X})$ with $\overleftarrow{X}, \overrightarrow{X} \in 2^Q, \sigma \in 2^\Sigma$ of a state $q \in Q$, it has *left successors* $\overleftarrow{X}$ and *right successors* $\overrightarrow{X}$. Thus, we can

get rid of the direction indicators -1 and 1. Note that we use 2^Σ in order to merge transitions that only differ in their actions. In standard notation, the transition $(\overleftarrow{X}, \sigma, \overrightarrow{X}) \in \delta(q)$ for any state $q \in Q$ is equivalent to

$$\forall\, s \in \sigma.\, \delta(q, s) \in \mathbb{B}^+\big((\overleftarrow{X} \times \{-1\}) \cup (\overrightarrow{X} \times \{1\})\big).$$

For simplicity, instead of defining runs of 2AAs over infinite trees, we are using forests, which are similar to trees with two directions, called *binary trees*. However, since the runs are defined on words rather than trees as input, forests give a much easier representation.

A run of a 2AA $\mathcal{A}$ on a word $\alpha \in \Sigma^\omega$ is defined using a *forest*

$$(V, E, \gamma_Q, \gamma_\mathbb{N})$$

where

- V indicates a *set of vertices*,
- $E \subseteq V \times V$ a *set of edges*,
- $\gamma_Q : V \to Q$ is the *state function*, and
- $\gamma_\mathbb{N} : V \to \mathbb{N}^{-1}$ is the *position function*.

Both functions are combined using $\gamma = \gamma_Q \times \gamma_\mathbb{N} : V \to Q \times \mathbb{N}^{-1}$. As its name suggests, the function $\gamma_\mathbb{N}$ indicates the position that $\mathcal{A}$ is currently reading on word α, that means

$$\forall\, v \in V. \forall\, (v, w) \in E.\, \gamma_\mathbb{N}(w) = \begin{cases} \gamma_\mathbb{N}(v) + 1, & \text{if } \mathcal{A} \text{ goes a step further on } \alpha \\ \gamma_\mathbb{N}(v) - 1, & \text{if } \mathcal{A} \text{ goes a step back on } \alpha. \end{cases}$$

Furthermore, given some vertex $v \in V$, we define some notations for our set of edges E:

- $E(v) = \{w \in V \mid (v, w) \in E\}$
- $E^+(v) = E(v) \cup E(E(v)) \cup \ldots$
- $\overleftarrow{E}(v) = \{w \in V \mid (\gamma_\mathbb{N}(w) = \gamma_\mathbb{N}(v) - 1\}$
- $\overrightarrow{E}(v) = \{w \in V \mid (\gamma_\mathbb{N}(w) = \gamma_\mathbb{N}(v) + 1\}$

Thus, a *run* of a 2AA $\mathcal{A}$ on a word $\alpha = \alpha_0\alpha_1 \ldots \in \Sigma^\omega$ is a forest $(V, E, \gamma_Q, \gamma_\mathbb{N})$ where the roots $\mathcal{R}$ satisfy the initial condition, i.e., $\forall\, r \in \mathcal{R}.\, \gamma(r) = (q_0, 0)$ with $q_0 \in Q_0$, and each vertex $v \in V$ satisfies the transition function, i.e.,

$$\big(\gamma_\mathbb{N}(v) = -1\big) \to E(v) = \emptyset,$$

or

$$\exists\, \sigma \in 2^{\Sigma}.\, \alpha_{\gamma_{\mathbb{N}}(v)} \in \sigma \text{ and } \big(\gamma_Q\big(\overleftarrow{E}(v)\big), \sigma, \gamma_Q(\overrightarrow{E}(v))\big) \in \delta\big(\gamma_Q(v)\big).$$

A run γ of a 2AA $\mathcal{A}$ is called *accepting* if $\forall\, v \in V.\, (\gamma_{\mathbb{N}}(v) = -1) \rightarrow \gamma_Q(v) \in Fin$, and any infinite branch is element in Acc.

A 2AA $\mathcal{A}$ with an alphabet Σ, a set of states Q and a transition function δ is called *very-weak* (2VWAA) if there exists a partial order $\preccurlyeq$ on Q such that

$$\forall\, q \in Q.\, \forall\, (\overleftarrow{X}, \sigma, \overrightarrow{X}) \in \delta(q).\, \forall\, q' \in \overleftarrow{X} \cup \overrightarrow{X}.\, q' \preccurlyeq q,$$

i.e., there is a straight line going through the runs of $\mathcal{A}$ without revisiting a state with a lower ranking.

A run $(V, E, \gamma_Q, \gamma_{\mathbb{N}})$ of a 2VWAA is called *progressing* if every infinite sequence $v_0 v_1 \ldots \in V^{\omega}$ satisfies

$$\forall\, N \geq 0.\, \exists\, i \geq 0.\, \gamma_{\mathbb{N}}(v_i) \geq N.$$

A run of a 2VWAA has a *loop* if two nodes on the same branch have the same label, formally

$$\exists\, v \in V.\, \exists\, w \in E^{+}(v).\, \gamma(v) = \gamma(w).$$

Otherwise, we call the run *loop-free*.

Thus, a 2AA is called *progressing* (respectively *loop-free*) if all runs of this automaton are progressing (respectively loop-free). This leads to two observations on a given 2AA $\mathcal{A}$:

1) Any loop-free run on $\mathcal{A}$ is progressing.

2) $\mathcal{A}$ is progressing iff it is loop-free.

Example. An example for a 2VWAA is illustrated in Figure 2.6. For the input word $\alpha = \{r_1\}\{r_2\}\emptyset\{g_1\}\{g_2\}\emptyset^{\omega} \in \Sigma^{\omega}$, there is additionally a run given as a forest depicted in Figure 2.7.

2.5.2 PLTL to Progressing 2VWAA

After introducing progressing 2VWAA in detail, we now present an algorithm for transforming a given PLTL formula into a progressing 2VWABA, i.e., with a Büchi acceptance condition. To do so, we assume the initial PLTL formula φ to be in NNF (compare Section 2.2.3). After defining some helper function, we give the mentioned tableau construction, followed by a theorem that it implies.

For the transformation, we first define several helper functions. The function

$$\otimes : 2^{2^Q \times 2^{\Sigma} \times 2^Q} \times 2^{2^Q \times 2^{\Sigma} \times 2^Q} \rightarrow 2^{2^Q \times 2^{\Sigma} \times 2^Q}$$

uses two outcomes of the transition function of a 2AA and outputs the merge of both. Thus, for $J_1, J_2 \in 2^{2^Q \times 2^\Sigma \times 2^Q}$,:

$$J_1 \otimes J_2 = \{\overleftarrow{X_1} \cup \overleftarrow{X_2}, \sigma_1 \cap \sigma_2, \overrightarrow{X_1} \cup \overrightarrow{X_2} \mid (\overleftarrow{X_1}, \sigma_1, \overrightarrow{X_1}) \in J_1, (\overleftarrow{X_2}, \sigma_2, \overrightarrow{X_2}) \in J_2\}.$$

Furthermore, for an arbitrary PLTL formula ψ, we define the function $\overline{\psi}$, which eliminates boolean connectives, inductively by

$$\begin{aligned} \overline{\psi} &= \{\{\psi\}\}, \text{ if } \psi \text{ is a temporal formula} \\ \overline{\psi_1 \vee \psi_2} &= \overline{\psi_1} \cup \overline{\psi_2} \\ \overline{\psi_1 \wedge \psi_2} &= \{X_1 \cup X_2 \mid X_1 \in \overline{\psi_1} \text{ and } X_2 \in \overline{\psi_2}\}. \end{aligned}$$

Before defining the construction, we introduce a special state. Since it is not trivial to observe a past modality at the beginning of a given infinite input word α, we use the additional state END. It is reached in a finite branch $v_0 v_1 \dots v_n \in V^*$ of an accepting run $(V, E, \gamma_Q, \gamma_\mathbb{N})$ of our automaton on α, if the current position $\gamma_\mathbb{N}(v_n)$ is outside of α, formally:

$$\gamma_Q(v_n) = \text{END} \Rightarrow \gamma_\mathbb{N}(v_n) = -1.$$

Construction 2. Given a PLTL formula φ in NNF on the set AP, let $\mathcal{A}_\varphi = \big(\Sigma, Q, Q_0, \delta, Fin, \text{Büchi}(F)\big)$ be the 2VWABA defined by

- $\Sigma = 2^{AP}$,
- $Q = TSub(\varphi) \cup AP \cup \overline{AP} \cup \{\text{END}\}$,
- $Q_0 = \overline{\varphi}$,
- $Fin = \{\text{END}\}$,
- F is the set of subformulas in $Q \setminus \{\text{END}\}$ that are not of the form $\varphi_1 \,\mathcal{U}\, \varphi_2$, and
- δ is defined below with Δ as its extension to $\mathbb{B}^+(Q)$:

$$
\begin{cases}
\delta(\mathit{false}) = \emptyset \\
\delta(\mathit{true}) = \{(\emptyset, \Sigma, \emptyset)\} \\
\delta(p) = \{(\emptyset, \Sigma_p, \emptyset)\} \text{ where } \Sigma_p = \{X \in \Sigma \mid p \in X\} \\
\delta(\neg p) = \{(\emptyset, \Sigma_{\neg p}, \emptyset)\} \text{ where } \Sigma_{\neg p} = \Sigma \setminus \Sigma_p \\
\delta(\bigcirc \psi) = \{(\emptyset, \Sigma, e) \mid e \in \overline{\psi}\} \\
\delta(\ominus \psi) = \{(e, \Sigma, \emptyset) \mid e \in \overline{\psi}\} \\
\delta(\circledcirc \psi) = \{(e, \Sigma, \emptyset) \mid e \in \overline{\psi}\} \cup \big\{(\{\text{END}\}, \Sigma, \emptyset)\big\} \\
\delta(\psi_1 \,\mathcal{U}\, \psi_2) = \Delta(\psi_2) \cup \big(\Delta(\psi_1) \otimes \big\{(\emptyset, \Sigma, \{\psi_1 \,\mathcal{U}\, \psi_2\})\big\}\big) \\
\delta(\psi_1 \,\mathcal{R}\, \psi_2) = \Delta(\psi_2) \otimes \big(\Delta(\psi_1) \cup \big\{(\emptyset, \Sigma, \{\psi_1 \,\mathcal{R}\, \psi_2\})\big\}\big) \\
\delta(\psi_1 \,\mathcal{S}\, \psi_2) = \Delta(\psi_2) \cup \big(\Delta(\psi_1) \otimes \big\{(\{\psi_1 \,\mathcal{S}\, \psi_2\}, \Sigma, \emptyset)\big\}\big) \\
\delta(\psi_1 \,\mathcal{T}\, \psi_2) = \Delta(\psi_2) \otimes \big(\Delta(\psi_1) \\
\qquad\qquad \cup \big\{(\{\psi_1 \,\mathcal{T}\, \psi_2\}, \Sigma, \emptyset), (\{\text{END}\}, \Sigma, \emptyset)\big\}\big) \\
\delta(\text{END}) = \emptyset
\end{cases}
$$

$$
\begin{cases}
\Delta(\psi) = \delta(\psi) \text{ if } \psi \text{ is a temporal formula} \\
\Delta(\psi_1 \vee \psi_2) = \Delta(\psi_1) \cup \Delta(\psi_2) \\
\Delta(\psi_1 \wedge \psi_2) = \Delta(\psi_1) \otimes \Delta(\psi_2).
\end{cases}
$$

Theorem 1. [12] *For any PLTL formula in negative normal form φ with n temporal subformulas, the automaton $\mathcal{A}_\varphi$ is a progressing two-way very-weak alternating automaton with at most $n+1$ states and $\mathcal{L}(\mathcal{A}_\varphi) = \varphi$.*

Its correctness is based on the definition of the PLTL operators, especially using the expansion laws (Equation 2.1). A_φ is very-weak by considering $\psi_1 \preccurlyeq \psi_2$ if $\psi_1 = \text{END}$ or if $\psi_1 \in \mathit{Sub}(\psi_2)$. Each infinite branch is strictly increasing from a certain node on it, resulting in the progressiveness.

Example. We apply the construction on formula ξ_2. Figure 2.6 shows the progressing 2VWABA for ξ_2.

2.5.3 Progressing 2VWAA to Transition-Based GBW

The idea of the construction of a transition-based Generalized Büchi word automaton out of a progressing 2VWAA $\mathcal{A}$ is to consider the sets $X_i = \gamma_q(\gamma_{\mathbb{N}}^{-1}(i))$ of an accepting run of the input automaton on some word $\alpha = \alpha_0\alpha_1\ldots$, and to let them build an accepting run on our new GBW $\mathcal{G}$. As an example, consider the 2VWABA $\mathcal{A}_{\xi_2}$ in Figure 2.6, and the accepting run of it on word $\alpha = \{r_1\}\{r_2\}\emptyset\{g_1\}\{g_2\}\emptyset^\omega \in \Sigma^\omega$, shown in Figure 2.7. The sets X_i are given by the states below the position indicators, e.g., $X_0 = \{q_0^1, q_1^1, q_0^2\}$ or $X_1 = \{q_0^1, q_1^1, q_0^2, q_1^2\}$.

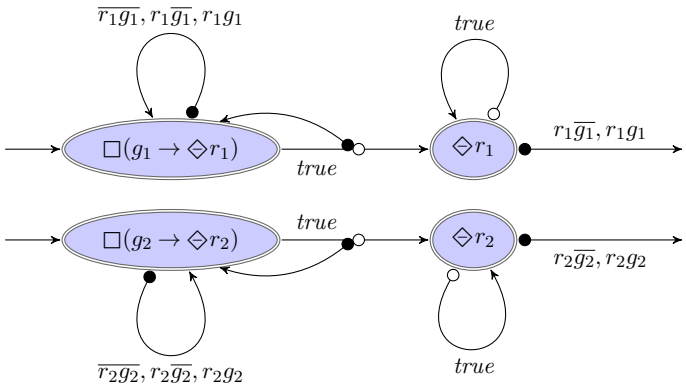


Figure 2.6: Progressing 2VWABA $\mathcal{A}_{\xi_2}$ for ξ_2 resulting from Construction 2. Filled circles denote right successor transitions, whereas empty circles denote left successors. Requests and grants are part of alphabet Σ. The set of directions only consists of the indicators for left and right.

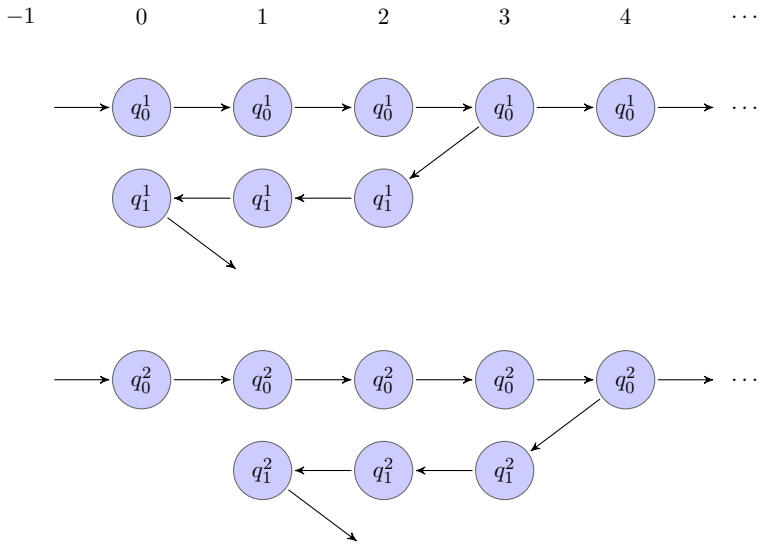


Figure 2.7: Run of 2VWABA $\mathcal{A}_{\xi_2}$ (Figure 2.6) on word $\alpha = \{r_1\}\{r_2\}\emptyset\{g_1\}\{g_2\}\emptyset^\omega \in \Sigma^\omega$. Numbers above denote the position $\gamma_{\mathbb{N}}$ on α. The labeling γ_Q can be taken directly from the vertices. We replaced the initial state names by q_0^1 and q_0^2, indicating the upper and lower state, respectively. Accordingly, the others are replaced by q_1^1 and q_1^2.

Algorithm 1 Saturation procedure of Construction 3.

```
    for each (X', σ, Z) ∈ ⨂_{q∈Y} δ(q)
(a) |  if X' ⊆ X
    |    if (Y, Z) ∉ ∇ then add (Y, Z) to ∇
    |    if (X, Y, Z, σ) ∉ δ' then add (X, Y, Z, σ) to δ'
    |  else
(b) |    for each (F, X, Y, σ') ∈ δ' with (F, X) ∈ Q'_0
    |      if (F, X ∪ X') ∉ ∇ then add (F, X ∪ X') to ∇
    |      add (F, X ∪ X') to Q'_0
(c) |    for each (V, W, X, σ''), (W, X, Y, σ') ∈ δ'
    |      if (W, X ∪ X') ∉ ∇ then add (W, X ∪ X') to ∇
    |      if (V, W, X ∪ X', σ'') ∉ δ' then
    |        add (V, W, X ∪ X', σ'') to δ'
```

Transitions of $\mathcal{G}$ should reflect the fact that each vertex v of a run on word α satisfies the transition function of $\mathcal{A}$, i.e., position $\gamma_{\mathbb{N}}(v) = i$ involves the sets $X_{i-1}, X_i, X_{i+1} \in 2^Q$, as well as the set of letters α_i. Thus, transitions of $\mathcal{G}$ are considered as four-tuples $(X_{i-1}, X_i, X_{i+1}, \alpha_i)$. In order to observe the classical representation of transitions as suggested in Section 2.4.1, one has to read this notation as $\delta'\big((X_{i-1}, X_i), \alpha_i\big) = (X_i, X_{i-1})$. This already indicates that states in our resulting GBW are given as pairs of sets of states, i.e., in $2^Q \times 2^Q$. Especially, the initial states will be of form $\{Fin\} \times 2^Q$. As explained before, the transitions are given as a set $\delta' \subseteq 2^Q \times 2^Q \times 2^Q \times 2^\Sigma$, which are equivalent to the representation $\delta' \subseteq (2^Q \times 2^Q) \times 2^\Sigma \times (2^Q \times 2^Q)$. Finally, the accepting set is given as a set of sets of transitions, i.e., $\mathcal{F}' \subseteq 2^{\delta'}$, such that, in each set $F' \in \mathcal{F}'$, at least one transition has to occur infinitely often. In the following, the construction and its corresponding theorem are presented.

Construction 3. For a progressing 2VWABA $\mathcal{A} = \big(\Sigma,\ Q,\ Q_0,\ \delta,\ Fin,\ \text{Büchi}(F)\big)$, we define the GBW $\mathcal{G}_\mathcal{A} = \big(\Sigma, Q', Q'_0, \delta', \text{GenBüchi}(\mathcal{F}')\big)$ by the following algorithm:

- *Initialization*: $Q'_0 = \{Fin\} \times Q_0$, $\nabla = \{Fin\} \times Q_0$, $\delta' = \emptyset$
- Apply *saturation procedure* for each $(X, Y) \in \nabla$ until fixed point is reached, i.e., no more changes occur. Its pseudo code can be observed in Algorithm 1.
- Finally, we set

$$
\begin{aligned}
- \ Q' &= \{(X,Y) \in 2^Q \times 2^Q \mid (X,Y) \in Q'_0 \text{ or} \\
&\qquad\qquad (X,Y) \text{ is reachable in } \delta'\} \\
- \ \mathcal{F}' &= \{T'_q \mid q \in Q \setminus (F \cup \{\text{END}\})\} \text{ where} \\
- \ T'_q &= \{(\overleftarrow{X}, X, \overrightarrow{X}, \sigma) \in \delta' \mid q \notin X \text{ or} \\
&\qquad \exists\, (\overleftarrow{Y}, \sigma', \overrightarrow{Y}) \in \delta(q) \text{ with } \overleftarrow{Y} \subseteq \overleftarrow{X}, \overrightarrow{Y} \subseteq \overrightarrow{X}, \sigma' \supseteq \sigma, q \notin \overrightarrow{Y}\}
\end{aligned}
$$

The idea of the saturation procedure can be split into the three parts (a), (b), and (c), as illustrated in Algorithm 1. In (a), the algorithm notices that (X', Y) is part of the already verified state (X, Y) and, therefore, it considers all possible successor states (Y, Z) and their underlying transitions. Otherwise, if (X', Y) is not part of the current state (X, Y), the algorithm checks all transitions going to state (X, Y). Thus, in part (b), it takes a look at all initial states build by X, and, based on the considered transition of Y, this state can also be build using the union of X and X'. Finally, in part (c), it analyzes all pairs of transitions that are ending in (X, Y), and, similar to the idea of part (b), it can also consider the union of X and X'.

Furthermore, there are multiple simplifications for our algorithm:

- *Saturation procedure*: In order to avoid redundant computations while searching for other changes, one could add integers, representing time stamps, to states and transitions, describing the number of iterations on the saturation procedure. Whenever a new element is added to ∇, 0 is assigned as its time stamp, whereas transitions added to δ' receive the current time stamp. Each time a pair $(X, Y) \in \nabla$ is considered for saturation, its time stamp is set to the current number of iterations, afterwards.

  Step (a) is only considered if the time stamp of (X, Y) is equal to 0. For step (b), we only consider transitions (F, X, Y, σ) where the time stamp is greater or equal to the pair (X, Y). For (c), we only consider pairs of transitions for which at least one time stamp is greater or equal to that of (X, Y).

- *Transition simplification*: It is possible to remove redundant transition on-the-fly. Given two transitions $(X, Y, Z, \sigma_1), (X, Y, Z, \sigma_2)$ with $\sigma_1 \subseteq \sigma_2$, the first transition can be deleted, because it is more restrictive.

- *State simplification*: After termination, two states $(X, Y), (X', Y')$ are called equivalent if both have the same outgoing transitions, i.e.,

  $$(X, Y, Z, \sigma) \in \delta' \Leftrightarrow (X', Y', Z, \sigma) \in \delta',$$

  and

  $$\forall\, q \in Q \setminus R.\, (X, Y, Z, \sigma) \in F'_q \Leftrightarrow (X', Y', Z, \sigma) \in F'_q.$$

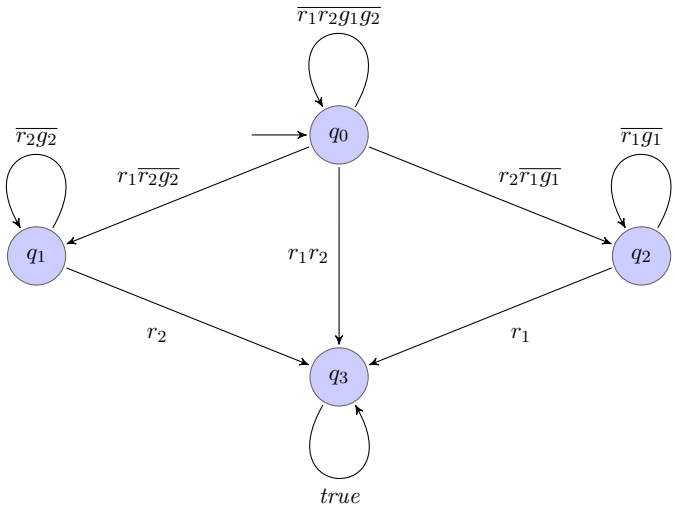


Figure 2.8: Transition-based GBW $\mathcal{G}'_{\xi_2}$ for ξ_2 that is equivalent to a solution for Construction Construction 3 for the automaton $\mathcal{A}_{\xi_2}$ in Figure 2.6. As there are no $\mathcal{U}$-subformulas in the underlying formula ξ_2, the set of sets of accepting transitions is empty. Thus, each infinite sequence in the automaton is part of its language.

> Equivalent states can be merged together and any transition that is pointing on one of the two states can be moved to the merged state.

Theorem 2. [12] *For any progressing two-way very-weak alternating Büchi automaton $\mathcal{A}$ with n states and f accepting states, the transition based Generalized Büchi word automaton $\mathcal{G}_\mathcal{A}$ accepts the same language, i.e., $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G}_\mathcal{A})$, with at most 2^n states and $n - f$ acceptance sets.*

Example. Since the resulting GBW $\mathcal{G}_{\mathcal{A}_{\xi_2}}$ for the progressing 2VWABA $\mathcal{A}_{\xi_2}$ in Figure 2.6 has to many states for a graphical representation, we just give a prefix of a run on it, based on the presented forest in Figure 2.7. Afterwards, we present an equivalent minimal transition-based GBW $\mathcal{G}'_{\mathcal{A}_{\xi_2}}$.

Thus, for the input word $\alpha = \{r_1\}\{r_2\}\emptyset\{g_1\}\{g_2\}\emptyset^\omega \in \Sigma^\omega$, a prefix of the run on $\mathcal{G}_{\mathcal{A}_{\xi_2}}$ is given by the sequence

$$(\{\text{END}\}, \{q_0^1, q_1^1, q_0^2\})\,(\{q_0^1, q_1^1, q_0^2\}, \{q_0^1, q_1^1, q_0^2, q_1^2\})\ldots$$

where each tuple represents a state in the GBW. In Figure 2.8, we give an equivalent minimal transition-based GBW $\mathcal{G}'_{\mathcal{A}_{\xi_2}}$ such that $\mathcal{L}(\mathcal{G}_{\mathcal{A}_{\xi_2}}) = \mathcal{L}(\mathcal{G}'_{\mathcal{A}_{\xi_2}})$.

2.5.4 Transition-Based GBW to NBW

As presented in [11], there is a construction from a transition-based Generalized Büchi word automaton with set of accepting sets $\mathcal{F} = \{F_1, \ldots, F_f\}$ to a non-deterministic Büchi word automaton, called *Round-Robin Construction* or *Counting Construction*. The idea behind it is to add a counter to each state, representing the current highest index i of the visited acceptance set F_i. Given index i, whenever there is taken another transition, the construction checks for all sets F_j with $i < j \leq f$ that are satisfied with this transition, and sets the next index to the highest visited j. For $i = f$, the counter starts again at 1. Thus, if we just visit states with counter f infinitely often, we ensure, that each acceptance set is visited infinitely often.

Construction 4. Given a transition-based GBW $\mathcal{G} = \big(\Sigma, Q, Q_0, \delta, \text{GenBüchi}(\mathcal{F})\big)$ with $\mathcal{F} = \{F_1, \ldots, F_f\}$ and $F_i \subseteq \delta'$ for each $i \in \{1, \ldots, f\}$ [4], we create the NBW $\mathcal{A} = (\Sigma, Q \times \{1, \ldots, f\}, Q_0 \times \{1\}, \delta', Q \times \{f\})$, where the transition function δ' is defined by

$$\delta'\big((q,i),\sigma\big) = \big\{(q', i') \mid q' \in \delta(q,\sigma) \text{ and } i' = \text{nxt}\big(i, (q,\sigma,q')\big)\big\}$$

with

$$\text{nxt}(i,t) = \begin{cases} \max\{i \leq j \leq f \mid \forall\, i < k \leq j.\, t \in F_k\}, & \text{if } i \neq f \\ \max\{1 \leq j \leq f \mid \forall\, 0 < k \leq j.\, t \in F_k\}, & \text{if } i = f \end{cases}$$

Theorem 3. [11] *For any transition-based Generalized Büchi word Automaton $\mathcal{G}$ with n states and f sets of accepting transitions, we can define a non-deterministic Büchi word automaton $\mathcal{A}$ with at most $n \cdot f$ states such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{A})$.*

Example. Since the set of accepting sets is empty for GBW $\mathcal{G}'_{\xi_2}$ in Figure 2.8, its NBW remains equal.

2.6 Bounded Synthesis for LTL

In the following, we introduce the approach of bounded synthesis for LTL by Finkbeiner and Schewe [9]. For this, we start with the general definition of the synthesis problem before explaining the idea of bounded synthesis. Afterwards, we present its solution by defining an annotation on run graphs in two different ways, by using either an emptiness game, or a constraint-based approach.

[4] As described, $(\overleftarrow{X}, X, \overrightarrow{X}, \sigma) \in \delta'$ is equivalent to the notation $\big((\overleftarrow{X}, X), \sigma, (X, \overrightarrow{X})\big) \in \delta'$ with $(\overleftarrow{X}, X), (X, \overrightarrow{X}) \in Q$.

2.6.1 The (Bounded) Synthesis Problem

The *synthesis problem* is given by the four-tuple

$$(I, O, i_0, \varphi)$$

where

- I is a set of boolean *input variables*,
- O is a set of boolean *output variables*,
- $i_0 \subseteq I$ is the fixed *initial input*, and
- φ is an LTL *specification* over the set of atomic propositions $AP = I \dot{\cup} O$.

We call specification φ *(finite-state) realizable* if there is an implementation as input-preserving 2^{AP}-labeled 2^I-transition system $\mathcal{T}$ that satisfies φ. The synthesis problem's goal is to decide whether φ is realizable.

To examine the possibly infinite search space of implementations in a structured way, [9] introduces the idea of *bounded synthesis*. In here, the set of implementations is considered up to a predefined bound. Assuming a guess $\mathcal{T} = (T, t_0, \tau, o)$ on the implementation for φ, it defines an annotation function on it that maps each state to a bounded domain. Thus, if the annotation is valid, it implies the satisfaction of the guess. Furthermore, the annotation can be used to reduce the problem to an emptiness check on safety automata. Finally, by giving an appropriate encoding as input to an SMT solver, it is possible to find a desired implementation along with its annotation function.

2.6.2 LTL to UCT

As a first step, we translate the specification φ into an equivalent UCT $\mathcal{U}_\varphi$. This takes several steps, resulting in an exponential blowup. The following briefly describes the construction as initially presented in [16].

Construction 5. Given an LTL formula φ, we first build its negation $\neg\varphi$. Afterwards, we translate $\neg\varphi$ into an equivalent NBW $\mathcal{A}_\varphi$ using, for example, Construction 2. Finally, we construct an UCT $\mathcal{U}_\varphi$ simulating $\mathcal{A}_\varphi$ along each path.

Example. We execute bounded synthesis on the example formula ξ_1. The underlying synthesis problem is defined by

$$(I = \{r_1, r_2\}, O = \{g_1, g_2\}, i_0 = \overline{r_1 r_2}, \varphi = \xi_1).$$

An NBW for $\neg\xi_1$, which is equivalent to the UCT for ξ_1, is given in a previous example, referencing to Figure 2.5.

2.6.3 Annotation on Run Graphs

A *run graph* of a tree automaton $\mathcal{A} = (\Sigma, \Upsilon, Q, Q_0, \delta, Acc)$ on a Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is a minimal directed graph $\mathcal{G} = (V, E)$ with vertices $V \subseteq Q \times T$ and edges $E \subseteq V \times V$, satisfying

- Each pair of initial states is a vertex in $\mathcal{G}$, i.e., $\forall\, q_0 \in Q_0.\, (q_0, t_0) \in V$, and
- Each vertex $(q, t) \in V$ satisfies the underlying transition relations, i.e.,

$$\forall\, (q,t) \in V.\, \big\{(q', \upsilon) \in Q \times \Upsilon \mid \big((q,t), (q', \tau(t,\upsilon)\big) \in E\big\} \text{ satisfies } \delta\big(q, o(t)\big).$$

We call a run graph *accepting* if every infinite path $(q_0, t_0)(q_1, t_1) \ldots \in V^\omega$ is part of the accepting condition, i.e., $q_0 q_1 \ldots \in Acc$.

Thus, given our guess as Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ and the UCT $\mathcal{U}_\varphi = (\Sigma, \Upsilon, Q, Q_0, \delta, \text{coBüchi}(C))$, we build the run graph $\mathcal{G} = (V, E)$ of $\mathcal{U}_\varphi$ on $\mathcal{T}$. Afterwards, we build an *annotation* $\lambda : V \to \{\text{␣}\} \cup \mathbb{N}$ on the run graph, indicating the maximal number of rejecting states occurring on some path to a node in the run graph. If λ only assigns natural numbers to the elements of $\mathcal{G}$, we talk about a *valid transition system*, implying an upper bound on the number of visits on the rejecting states. Furthermore, the set of all transition systems with valid annotations is equal to those that are accepted by the automaton, and, thus, implementing φ.

For terminology, we call annotation λ *c-bounded* if $\lambda : V \to \{\text{␣}\} \cup \{0, \ldots, c\}$, and *bounded* if it is c-bounded for some $c \in \mathbb{N}$. An annotation is *valid* if it satisfies the following conditions:

- The initial vertices are annotated by a natural number, formally, $\forall\, q_0 \in Q_0.\, \lambda(q_0, t_0) \neq \text{␣}$.
- Each path of $\mathcal{G}$ has an increasing annotation, which is strictly increasing whenever another rejecting state is encountered. Formally, for some vertex $(q, t) \in V$ and $(q', \upsilon) \in \delta(q, o(t))$, we define

$$\lambda(q,t) \begin{cases} < \lambda\big(q', \tau(t,\upsilon)\big), & \text{if } q' \in \mathcal{C} \\ \leq \lambda\big(q', \tau(t,\upsilon)\big), & \text{otherwise.} \end{cases}$$

The following theorem states that the existence of a finite transition system with a valid annotation is a *sufficient condition* for the realizability of specification φ.

Theorem 4. [9] *A finite-state Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is accepted by a universal co-Büchi tree automaton $\mathcal{U} = \big(\Sigma, \Upsilon, Q, Q_0, \delta, \text{coBüchi}(C)\big)$ if, and only if, it has a valid $(|T| \cdot |C|)$-bounded annotation.*

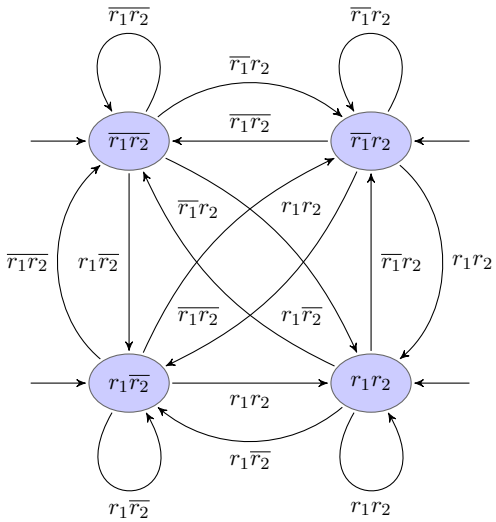


Figure 2.9: DST $D_{\mathcal{I}}$ for set of inputs $I = \{r_1, r_2\}$. Since every state of it reflects the previous input, it is used to check input-preservation.

Subsequently, its existence is a necessary condition, defining a bound on the number of states of implementation $\mathcal{T}$.

Corollary 1. [9] *If a universal co-Büchi tree automaton* $\mathcal{U} = \big(\Sigma, \Upsilon, Q, Q_0, \delta, \mathit{coBüchi}(C)\big)$ *accepts an input-preserving transition system, then* $\mathcal{U}$ *also accepts a finite input-preserving transition system* $\mathcal{T} = (T, t_0, \tau, o)$ *with* $|T| = |Q|!^2 \cdot |\mathcal{I}|$ *where* $\mathcal{I} = 2^I$*. Thus,* $\mathcal{T}$ *has a valid* $(|T| \cdot |C|) = (|Q|!^2 \cdot |\mathcal{I}| \cdot |C|)$*-bounded annotation.*

For its correctness, they first translate $\mathcal{U}_\varphi$ into an equivalent DPT $\mathcal{P}$ with $n!^2$ states and $2n$ colors [22, 26]. The input-preservation is checked by a DST $D_{\mathcal{I}}$, whose states are formed by the inputs $\mathcal{I}$. In every state $i \in \mathcal{I}$, $D_{\mathcal{I}}$ checks if i agrees with the given input, and sends the successor input $i' \in \mathcal{I}$ into direction i'. An example for $I = \{r_1, r_2\}$ can be observed in Figure 2.9. Finally, we evaluate the emptiness game of the product automaton of $\mathcal{P}$ and $\mathcal{D}_I$.

Example. Figure 2.10 shows the run graph of the UCT in Figure 2.5 on the implementation in Figure 2.2 for formula ξ_1.

2.6.4 Bounded Synthesis by Emptiness Check

Based on the given annotation on our transition system and the idea of the sufficient bound, we can reduce the synthesis problem to an emptiness check on safety automata. The following theorem describes the construction of a family of DSTs up to one of the previously defined bounds.

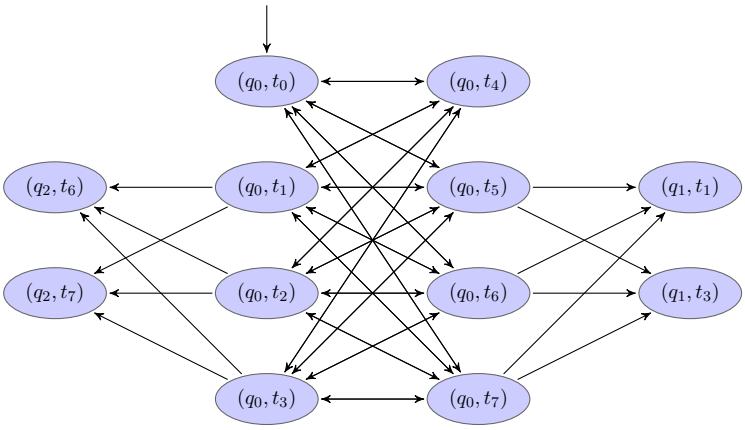


Figure 2.10: Run graph of UCT $\mathcal{U}_{\xi_1}$ in Figure 2.5 on TS $\mathcal{T}_{\xi_1}$ in Figure 2.2. Since our implementation is finite, we are able to define at most a c-bounded annotation on the run graph with $c = |T| \cdot |C| = 8 \cdot 2 = 16$. We define a 1-bounded valid annotation with $\lambda(q_0, \cdot) = 0$, and 1 for all other vertices

Theorem 5. *Given a universal co-Büchi tree automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, Q_0, \delta, \text{coBüchi}(C))$, we can construct a family of deterministic safety tree automata $\{D_c = (\Sigma, \Upsilon, S_c, s_0, \delta_c) \mid c \in \mathbb{N}^+\}$ such that the automaton D_c accepts a transition system $\mathcal{T}$ if, and only if, $\mathcal{T}$ has a valid $(b \cdot |C|)$-bounded annotation, where b is one of the previously defined bounds.*

The DST $D = (\Sigma, \Upsilon, S_c, s_0, \delta_c)$ for $c \in \{1, \ldots, b \cdot |C|\}$ is constructed as follows. Note that, if we do not state the safety condition explicitly, it simply requires to only have infinite branches on each run.

- *State space* S_c describes the set of functions $\{Q \to \{\llcorner\} \cup \{1, \ldots, c\}\}$ such that each state $s \in S$ indicates how many times a rejecting state in C may have been visited on some trace of the run graph $\mathcal{G}$ that passes the current position in the TS. The limitation on the number of visits is based on its index c.

- The *initial function* $s_0 \in S$ maps 0 to the initial states of $\mathcal{U}_\varphi$ and $\llcorner$ to all remaining, formally, for some state $q \in Q$,

$$s_0(q) = \begin{cases} 0, & \text{if } q \in Q_0 \\ \llcorner, & \text{for all } q \in Q \setminus Q_0. \end{cases}$$

- For the *transition function* δ_c, we first define another helper function

$$\delta_c^+ : S_c \times \Sigma \to 2^{(Q \times \{\llcorner\} \cup \{1, \ldots, c\}) \times \Upsilon}$$

that records, for all directions $\upsilon \in \Upsilon$ and successor states $q' \in Q$, the new number of visits of rejecting states, i.e.,

$$\delta_c^+(s,\sigma) = \left\{ \Big(\big(q', s(q)+f(q')\big), \upsilon \Big) \;\Big|\; q, q' \in Q, s(q) \neq \text{␣}, (q', \upsilon) \in \delta(q,\sigma) \right\}$$

$$\text{with } f(q) = \begin{cases} 1, & \forall q \in F \\ 0, & \forall q \notin F \end{cases} \quad \text{collects the transitions of } \mathcal{U}.$$

Note that the sum of $s(q)$ and $f(q')$ can not go out of bound and, thus, it is possible to have dead-ends within our automaton. Visiting these states would violate the safety condition.

The transition function $\delta_c : S_c \times \Sigma \to \mathbb{B}^+\big((Q \to \{\text{␣}\} \cup \{1,\ldots,c\}) \times \Upsilon\big)$ is then defined as conjunction of all encountered numbers of visits, divided by the set of directions. Formally,

$$\delta_c(s,\sigma) = \bigwedge_{\upsilon \in \Upsilon} (s_\upsilon, \upsilon)$$

with $\max\{\emptyset\} = \text{␣}$, and

$$s_\upsilon(q) = \max\big\{ n \in \{1,\ldots,c\} \mid \big((q,n),\upsilon\big) \in \delta_c^+(s,\sigma) \big\}.$$

After this, we are able to produce the *emptiness game* for D_i with $i \in \{1,\ldots,c\}$ in product with automaton D_I using the initial input i_0. Such an emptiness game consists of two players, player *accept* and player *reject*. In her states, the respective player chooses one of the given successor states.

For our emptiness game, player reject chooses some input I, such that player accept has to answer by grants. The game is won by player accept if her defined *strategy* lets each play on the game be infinite. Otherwise, player reject wins.

Example. For the initial input i_0, the respective emptiness game for the product automaton of D_1 together with D_I (Figure 2.9) is presented in Figure 2.11.

2.6.5 Bounded Synthesis using SMT Solver

Furthermore, an alternative constraint-based approach for bounded synthesis using an SMT solver is presented. Instead of guessing the desired input-preserving transition system, implementation and annotation are encoded as uninterpreted functions in first-order logic modulo finite integer arithmetic. Thus, bounded synthesis is reduced to the satisfiability of our encodings as input of an SMT solver.

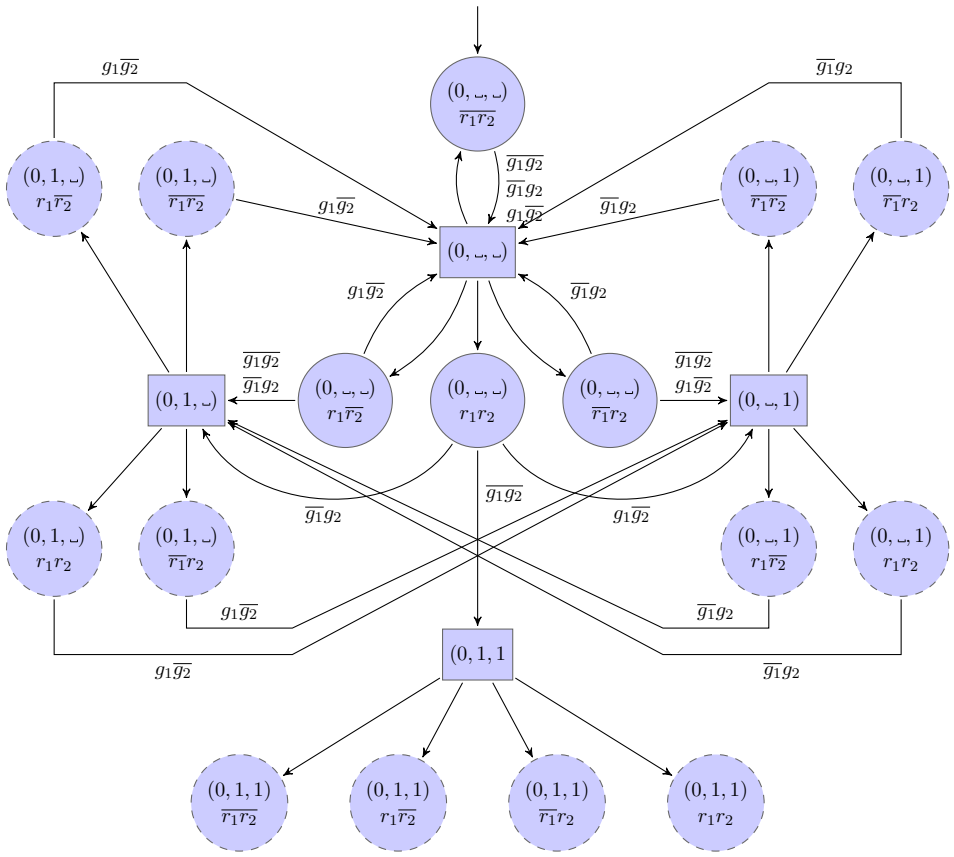


Figure 2.11: Respective emptiness game for the product automaton of DST D_1 together with DST D_I (Figure 2.9). Circles denote states of player accept, rectangles those of player reject. Some states of player accept are dashed, meaning that they are not completely expanded, i.e., for the given parameter $c = 1$, these transitions would be out of bound. The system player wins by avoiding the visit to player reject's state $(0, 1, 1)$.

To construct our constraint system, we assume the existence of a UCT $\mathcal{U}_\varphi = \big(2^{AP}, 2^I, Q, Q_0, \delta, \text{coBüchi}(C)\big)$ describing input specification φ over AP. We want to specify the existence of a finite input-preserving 2^{AP}-labeled 2^I-transition system $\mathcal{T} = (T, t_0, \tau, o)$ that is accepted by $\mathcal{U}_\varphi$ and, furthermore, has a valid annotation λ. This is done in multiple steps:

1. *Encode transition function τ*

   For each direction $\upsilon \in 2^I$, we introduce a unary function symbol $\tau_\upsilon : T \to T$ that maps each state $t \in T$ to its successor on direction υ, formally, $\tau_\upsilon(t) = \tau(t, \upsilon)$.

2. *Encode labeling function o*

   In order to decide if a state $t \in T$ is labeled with proposition $a \in AP$, we introduce for each proposition a unary predicate symbol $a : T \to \mathbb{B}$. It maps a state to *true* if its underlying proposition is part of the state, i.e., $a(t) = \mathit{true}$ iff $a \in o(t)$.

3. *Encode annotation λ*

   For our annotation, we introduce for each state $q \in Q$ of UCT $\mathcal{U}_\varphi$ two unary symbols:

   - $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$ where it maps state $t \in T$ to *true* if, and only if, its annotation is a natural number, formally, $\lambda_q^{\mathbb{B}}(t) = \mathit{true}$ iff $\lambda(q, t) \in \mathbb{N}$.
   - $\lambda_q^{\#} : T \to \mathbb{N}$ where it maps state $t \in T$ to its annotation value $\lambda(q, t)$, or is undefined if $\lambda(q, t) = \text{␣}$, formally:

   $$\lambda_q^{\#}(t) = \begin{cases} \lambda(q,t) & \text{if } \lambda(q,t) \in \mathbb{N} \\ \text{undef.} & \text{if } \lambda(q,t) = \text{␣}. \end{cases}$$

4. *Formalization of valid annotation for $\mathcal{T}$*

   As a recap, we call an annotation valid if it only assigns natural numbers to the vertices in the run graph. Thus, we have to ensure that successor v' of vertex v is assigned a natural number whenever v is. The assignment is strictly greater if v' contains a final state of F. We define the following abbreviations:

   - For some state $t \in T$, the symbol $\overrightarrow{a} : T \to 2^{AP}$ represents the labeling of t, formally, $\overrightarrow{a}(t) = o(t)$.
   - For some state $q \in Q$, the symbol $\rhd_q$ gives the correct greater sign, i.e., $\rhd_q$ is $>$ if $q \in F$, and $\geq$ otherwise.

Thus, in order to achieve a valid annotation of $\mathcal{T}$, we formalize the constraint

$$\forall\, t.\, \lambda_q^{\mathbb{B}}(t) \wedge (q', \upsilon) \in \delta\big(q, \overrightarrow{a}(t)\big) \rightarrow \lambda_{q'}^{\mathbb{B}}\big(\tau_\upsilon(t)\big) \wedge \lambda_{q'}^{\#}\big(\tau_\upsilon(t)\big) \rhd_{q'} \lambda_q^{\#}(t).$$

Additionally, we require that all pairs of initial states are initially labeled by a natural number (w.l.o.g. 0), formally, $\forall\, q_0 \in Q_0.\, \lambda_{q_0}^{\mathbb{B}}(t_0)$.

5. *Guarantee input-preservation of $\mathcal{T}$*

   By the definition of input-preserving, we have to guarantee that the label of each state $t \in T$ reflects the previous input. Therefore, for each input symbol $a \in I$ and each direction $\upsilon \in 2^I$, we have to specify whether $a \in \upsilon$. We add the constraints

   - $\forall\, t.\, a\big(\tau_\upsilon(t)\big)$ if $a \in \upsilon$, and
   - $\forall\, t.\, \neg a\big(\tau_\upsilon(t)\big)$ if $a \notin \upsilon$.

6. *Initial labeling of t_0*

   The initial state is labeled with the initial input i_0, i.e., we build the conjunction $p_1(t_0) \wedge \ldots \wedge p_n(t_0)$ with $i_0 = \{p_1, \ldots, p_n\}$.

Such a constraint system, resulting from a given LTL specification, is satisfiable (modulo a theory with order) if, and only if, the LTL specification is finite-state realizable. Furthermore, given UCT $\mathcal{U}_\varphi$, the constraint system is of size

$$O(|\delta| \cdot |AP| + |I| \cdot 2^{|I|}).$$

By completely resolving the universal quantification of $\mathcal{U}_\varphi$, the size of the constraint system additionally depends on the size $b_\mathcal{T}$ of its transition system $\mathcal{T}$, leading to size

$$O\big(b_\mathcal{T} \cdot (|\delta| \cdot |AP| + |I| \cdot 2^{|I|})\big).$$

Example. Given the UCT $\mathcal{U}_{\xi_1} = \big(\Sigma, \Upsilon, Q, Q_0, \delta, \text{Büchi}(F)\big)$ in Figure 2.5 with $\Sigma = 2^{\{r_1, r_2, g_1, g_2\}}$ and $\Upsilon = 2^{\{r_1, r_2\}}$, we specify the existence of a finite input-preserving Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ for formula ξ_1. We follow the previously introduced ordering.

1. *Encode transition function τ*

   For each direction $\upsilon \in \{\overline{r_1}\,\overline{r_2}, \overline{r_1} r_2, r_1 \overline{r_2}, r_1 r_2\}$, we introduce the unary function symbols $\tau_\upsilon : T \rightarrow T$, e.g., $\tau_{\overline{r_1}\,\overline{r_2}} : T \rightarrow T$.

2. *Encode labeling function o*

   For each proposition $p \in \{r_1, r_2, g_1, g_2\}$, we introduce the unary predicate symbol $p : T \rightarrow \mathbb{B}$, e.g., $r_1 : T \rightarrow \mathbb{B}$.

3. *Encode annotation λ*

   For each state $q \in \{q_0, q_1, q_2\}$, in the following abbreviated by their indices, we introduce unary predicate symbol $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$, e.g., $\lambda_0^{\mathbb{B}} : T \to \mathbb{B}$, as well as unary function symbol $\lambda_q^{\#} : T \to \mathbb{N}$, e.g., $\lambda_0^{\#} : T \to \mathbb{N}$.

4. *Formalization of valid annotation for $\mathcal{T}$*

   We encode each transition of $\mathcal{U}_{\xi_1}$ separately. (a) encodes the self-loop of q_0, (b) the forbidden transition with both grants given simultaneously. (c) and (d) describe the remaining transitions of q_0, and (e) and (f) the remaining self-loops of q_1, q_2.

   (a) $\forall t.\, \lambda_0^{\mathbb{B}}(t) \to \lambda_0^{\mathbb{B}}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \geq \lambda_0^{\#}(t)$
   $\wedge\, \lambda_0^{\mathbb{B}}\big(\tau_{\overline{r_1}r_2}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{\overline{r_1}r_2}(t)\big) \geq \lambda_0^{\#}(t)$
   $\wedge\, \lambda_0^{\mathbb{B}}\big(\tau_{r_1\overline{r_2}}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{r_1\overline{r_2}}(t)\big) \geq \lambda_0^{\#}(t)$
   $\wedge\, \lambda_0^{\mathbb{B}}\big(\tau_{r_1r_2}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{r_1r_2}(t)\big) \geq \lambda_0^{\#}(t)$

   (b) $\forall t.\, \lambda_0^{\mathbb{B}}(t) \to \neg g_1(t) \vee \neg g_2(t)$

   (c) $\forall t.\, \lambda_0^{\mathbb{B}}(t) \wedge r_1(t) \wedge \neg g_1(t) \to \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1}r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1}r_2}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{r_1\overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1\overline{r_2}}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{r_1r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1r_2}(t)\big) > \lambda_0^{\#}(t)$

   (d) $\forall t.\, \lambda_0^{\mathbb{B}}(t) \wedge r_2(t) \wedge \neg g_2(t) \to \lambda_2^{\mathbb{B}}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{\overline{r_1}r_2}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{\overline{r_1}r_2}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{r_1\overline{r_2}}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{r_1\overline{r_2}}(t)\big) > \lambda_0^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{r_1r_2}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{r_1r_2}(t)\big) > \lambda_0^{\#}(t)$

   (e) $\forall t.\, \lambda_1^{\mathbb{B}}(t) \wedge \neg g_1(t) \to \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1}r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1}r_2}(t)\big) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{r_1\overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1\overline{r_2}}(t)\big) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}\big(\tau_{r_1r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1r_2}(t)\big) > \lambda_1^{\#}(t)$

   (f) $\forall t.\, \lambda_2^{\mathbb{B}}(t) \wedge \neg g_2(t) \to \lambda_2^{\mathbb{B}}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{\overline{r_1}\overline{r_2}}(t)\big) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{\overline{r_1}r_2}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{\overline{r_1}r_2}(t)\big) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{r_1\overline{r_2}}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{r_1\overline{r_2}}(t)\big) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}\big(\tau_{r_1r_2}(t)\big) \wedge \lambda_2^{\#}\big(\tau_{r_1r_2}(t)\big) > \lambda_2^{\#}(t)$

   Furthermore, the pair of initial states (q_0, t_0) is initialized with a natural number (w.l.o.g. 0), i.e., $\lambda_0^{\mathbb{B}}(0)$.

5. *Guarantee input-preservation of $\mathcal{T}$*

   We specify for each input $r \in \{r_1, r_2\}$ whether it is element of direction $\upsilon \in 2^{\{r_1, r_2\}}$. For the four possible directions, this leads to the following

constraint.

$$\forall t.\, r_1(\tau_{r_1 r_2}(t)) \wedge r_2(\tau_{r_1 r_2}(t)) \wedge r_1(\tau_{r_1 \overline{r_2}}(t)) \wedge \neg r_2(\tau_{r_1 \overline{r_2}}(t)) \\ \neg r_1(\tau_{\overline{r_1} r_2}(t)) \wedge r_2(\tau_{\overline{r_1} r_2}(t)) \wedge \neg r_1(\tau_{\overline{r_1 r_2}}(t)) \wedge \neg r_2(\tau_{\overline{r_1 r_2}}(t))$$

6. *Initial labeling of* t_0

   The initial input is set by the constraint $\neg r_1(0) \wedge \neg r_2(0)$.

Building the conjunction of these constraints and handling it as input of an SMT solver leads to a solution for bounded synthesis of ξ_1, i.e., an implementation $\mathcal{T}$ for ξ_1 as input-preserving finite transition system, as well as a valid annotation on $\mathcal{T}$ for an appropriate bound.

Chapter 3

The Naive Approach: Gabbay's Separation Theorem

Whenever a paper is arguing about PLTL, it refers to the approach of Gabbay, presented in [10]. His famous *Separation Theorem* gives various rewriting rules for pushing past-operators out of the scope of future-operators, and vice versa. We presented its main characteristics, as well as drawbacks, in Section 2.2.5. Now, we want to use his theorem in order to obtain a formula that is initially equivalent to our input, and, thus, allows for the execution of standard bounded synthesis. We start with a quick insight into a rewriting of his rules. We justify this rewriting by the outdated notation of PLTL, as used in the theorem, combined with strict operators only. For completeness, we furthermore present the complete rewriting, as well as the idea of the correctness proof in partial detail, in Appendix A. Afterwards, we examine its combination with the bounded synthesis approach and give an argumentation why it is not usable in practice. We assume the synthesis problem

$$(I, O, i_0, \varphi)$$

where input and output variables are disjunct and their union builds the set of atomic propositions, i.e., $AP = I \dot{\cup} O$. The formula φ is a PLTL formula over AP, built only using standard operators, i.e., neither boolean (except conjunction) nor temporal abbreviations. This allows us to observe φ without any other normal form.

3.1 Gabbay's Separation Theorem

Theorem 6. (**Separation Theorem**) [10] *Given some PLTL formula φ, there exists an initially equivalent LTL formula φ', notated $\varphi \equiv_i \varphi'$, that is*

obtained by a series of rewriting rules applied on formula φ, such that φ' gives a clear separation of past, present, and future formulas, combined by boolean connectives. Formally,

$$\varphi \equiv_i \varphi' \in \mathbb{B}^+(\varphi_{past}, \varphi_{present}, \varphi_{future}),$$

such that φ_{past} and φ_{future} are temporal formulas in pure-past and pure-future, respectively, and $\varphi_{present}$ describes an assertion.

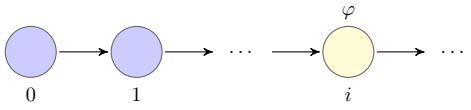
We distinguish the following 14 cases for pushing future-operators out of the scope of past-operators. Based on the time-equivalence of future and past, one can mirror the cases with reversed operators to obtain the other direction. Let φ_1 and φ_2 be some arbitrary formulas, as well as a and b.

1. $a \,\mathcal{S}\, \big(b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
2. $\big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, b$
3. $a \,\mathcal{S}\, \big(b \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
4. $\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, b$
5. $\big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
6. $\big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
7. $\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
8. $\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$
9. $\ominus(\bigcirc \varphi_1)$
10. $\ominus(\varphi_1 \,\mathcal{U}\, \varphi_2)$
11. $a \,\mathcal{S}\, (b \wedge \bigcirc \varphi_1)$
12. $(a \vee \bigcirc \varphi_1) \,\mathcal{S}\, b$
13. $(a \vee \bigcirc \varphi_1) \,\mathcal{S}\, (b \wedge \bigcirc \varphi_1)$

Note that, besides the difference at the initial position, the operators $\ominus$ and $\circleddash$ are handled in the same way. Thus, we only observe the operator $\ominus$ in our case distinction. At the end of the elimination for some arbitrary subformula ψ, each outer-most formula $\ominus\psi$ is mapped to *false*, whereas each outer-most formula $\circleddash\psi$ is mapped to *true*.

This thesis presents each elimination $1-14$ in detail by giving a graphical explanation, as well as the appropriate equivalent rewriting. In this chapter, we only want to give insights into our reasoning, that is based on the original approach of Gabbay. Therefore, we present the intuitive Elimination 1, as well as the complicated rules for Elimination 2 using double-negation.

For now, we assume each formula φ to be satisfied at a position $i \in \mathbb{N}$ of a given word $\alpha = \alpha_0\alpha_1 \ldots$, i.e., $\alpha_i \vDash \varphi$. The following diagram shows a representation of this situation. Further explanations will do an unfolding and a case distinction of φ at position i.

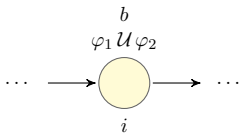


For any position $i, j, k \in \mathbb{N}$, we assume $k < j < i$.

Elimination 1: $\varphi = a\,\mathcal{S}\big(b \wedge (\varphi_1\,\mathcal{U}\,\varphi_2)\big)$

We distinguish the following four cases:

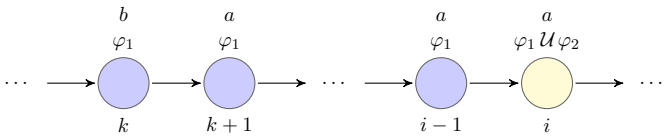
1) Right side is satisfied **at** i:



Formula:

$$\begin{aligned}
\psi_1 &= b \wedge (\varphi_1\,\mathcal{U}\,\varphi_2) \\
&\overset{2.1}{=} b \wedge \Big(\varphi_2 \vee \big(\varphi_1 \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)\big)\Big) \\
&\overset{\text{Dist.}}{=} \underbrace{\big(b \wedge \varphi_2\big)}_{\psi_1'} \vee \underbrace{\big(b \wedge \varphi_1 \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)\big)}_{\psi_1''}
\end{aligned}$$

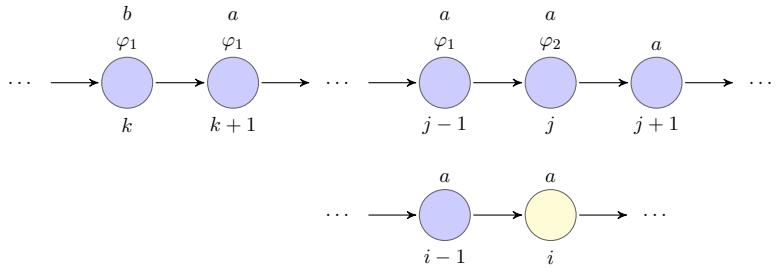
2) Right side is satisfied **before** i, φ_2 holds **at / after** i:



Formula:

$$\begin{aligned}
\psi_2 &= \ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \big(\varphi_1\,\mathcal{U}\,\varphi_2\big) \\
&\overset{2.1}{=} \ominus\Big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\Big) \wedge a \wedge \Big(\varphi_2 \vee \big(\varphi_1 \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)\big)\Big) \\
&\overset{\text{Dist.}}{=} \underbrace{\Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_2\Big)}_{\psi_2'} \\
&\quad \vee \underbrace{\Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_1 \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)\Big)}_{\psi_2''}
\end{aligned}$$

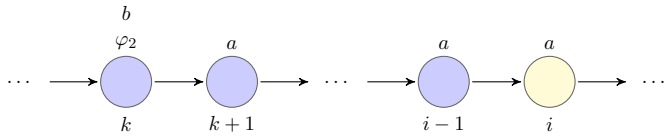
3) Right side is satisfied **before** i, φ_2 holds **before** i **after** k:



Formula:

$$\psi_3 = a \,\mathcal{S}\, \Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_2\Big)$$

4) Right side is satisfied **before** i, φ_2 holds **before** i **at** k:



Formula:

$$\psi_4 = a \,\mathcal{S}\, (b \wedge \varphi_2)$$

It should be clear that these cases are covering all possibilities. Furthermore, we are also aware of the fact that our distinctions are covering some cases twice. To deal with this, we made previous transformations in order to combine them:

$$\begin{aligned}
\varphi \equiv_i \varphi' &= \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \\
&= \psi_1' \vee \psi_1'' \vee \psi_2' \vee \psi_2'' \vee \psi_3 \vee \psi_4 \\
&= \underbrace{(\psi_1'' \vee \psi_2'')}_{\psi_1^f} \vee \underbrace{(\psi_2' \vee \psi_3)}_{\psi_2^f} \vee \underbrace{(\psi_1' \vee \psi_4)}_{\psi_3^f}
\end{aligned}$$

with

- $$\begin{aligned}
\psi_1^f &= \psi_1'' \vee \psi_2'' \\
&= \Big(b \wedge \varphi_1 \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)\Big) \\
&\vee \Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_1 \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)\Big) \\
&\overset{\text{Dist.}}{=} \Big(\Big(b \wedge \varphi_1\Big) \vee \Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_1\Big)\Big) \wedge \bigcirc\Big(\varphi_1 \,\mathcal{U}\, \varphi_2\Big) \\
&\overset{2.1}{=} \big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge \bigcirc\big(\varphi_1 \,\mathcal{U}\, \varphi_2\big)
\end{aligned}$$

- $\psi_2^f = \psi_2' \vee \psi_3$
  $= \Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_2\Big)$
  $\vee \Big(a\,\mathcal{S}\,\Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_2\Big)\Big)$
  $\overset{\text{Def.}}{=} a\,\mathcal{S}\,\Big(\ominus\big((a \wedge \varphi_1)\,\mathcal{S}\,(b \wedge \varphi_1)\big) \wedge a \wedge \varphi_2\Big)$
- $\psi_3^f = \psi_1' \vee \psi_4$
  $= \big(b \wedge \varphi_2\big) \vee \big(a\,\mathcal{S}\,(b \wedge \varphi_2)\big)$
  $\overset{\text{Def.}}{=} a\,\mathcal{S}\,(b \wedge \varphi_2)$

Elimination 2: $\varphi = \big(a \vee (\varphi_1\,\mathcal{U}\,\varphi_2)\big)\,\mathcal{S}\,b$

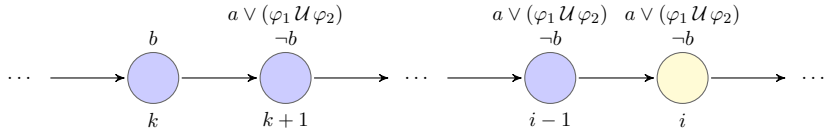
Again, we distinguish several cases:

1) For the easiest case, b holds **at** position i.

   Formula:

   $$\psi_1 = b$$

Thus, for the remaining cases, we can assume that b is satisfied **before** i.



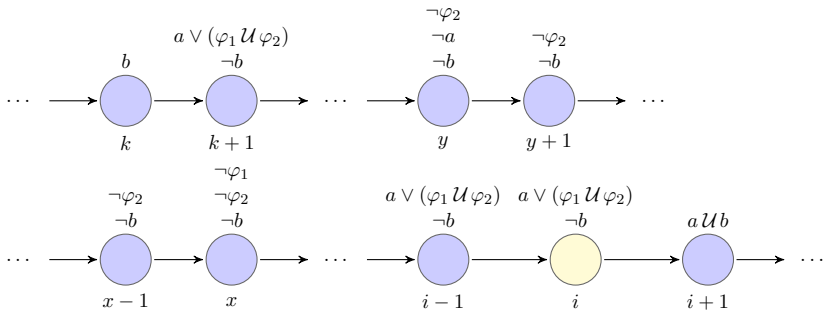
To make a statement about the future, we distinguish whether $\bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)$ holds at position i, i.e., whether the $\mathcal{U}$-statement is possibly continued in position $i+1$.

2) b holds **before** i, $\bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)$ is **satisfied** at i

   For this, we observe the negative case, i.e., there is some position $y \in \mathbb{N}$ with $k > y \geq i$ such that neither a nor $\varphi_1\,\mathcal{U}\,\varphi_2$ are satisfied at y. Formally:

$$\begin{aligned}
\neg\big(a \vee (\varphi_1\,\mathcal{U}\,\varphi_2)\big) &= (\neg a) \wedge \neg(\varphi_1\,\mathcal{U}\,\varphi_2) \\
&\overset{\text{Def.}}{=} (\neg a) \wedge (\neg\varphi_1\,\mathcal{R}\,\neg\varphi_2) \\
&\overset{2.2}{=} (\neg a) \wedge \Big((\Box\neg\varphi_2) \vee \big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big) \\
&\overset{\text{Dist.}}{=} \underbrace{\Big((\neg a) \wedge (\Box\neg\varphi_2)\Big)}_{\psi'} \vee \underbrace{\Big((\neg a) \wedge \big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big)}_{\psi''}
\end{aligned}$$

As marked, this triggers two possibilities ψ' and ψ'', where the first describes that $\neg\varphi_2$ holds forever, whereas, for the other, $\neg\varphi_2$ is eventually answered by $\neg\varphi_1$. Based on our assumption at position i, the first case is not possible. Thus, there is some position $x \in \mathbb{N}$ with $y \leq x \leq i$, such that the right side of the $\mathcal{U}$-statement of ψ'' holds at x. Graphically:



We build a formula κ describing the situation at position x:

$$\sigma_x \vDash \big((\neg b \wedge \neg\varphi_2)\, \mathcal{S}\, (\neg a \wedge \neg b \wedge \neg\varphi_2)\big) \wedge \neg\varphi_1 = \kappa$$

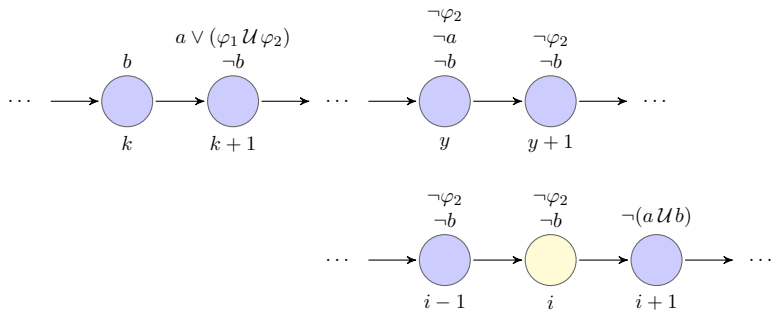
κ implies a position $k < y \leq x$ such that $\sigma_y \vDash \neg a \wedge \neg(\varphi_1 \mathcal{U} \varphi_2)$. Therefore, by negating this formula, we prevent the described situation, defining our second case.

Formula:

$$\psi_2 = \big((\neg b \wedge \neg\kappa)\, \mathcal{S}\, b\big) \wedge \bigcirc(\varphi_1 \mathcal{U} \varphi_2)$$

3) b holds **before** i, $\bigcirc(\varphi_1 \mathcal{U} \varphi_2)$ is **not satisfied** at i

In here, based on the presented cases ψ' and ψ'', position $i+1$ would not continue an open answer for an $\varphi_1 \mathcal{U} \varphi_2$. Thus, we have to prevent to start a new request for φ_2 by preventing φ_1 at position i. This is done by including a slightly modified version of κ at i. The negative situation, which we are going to prevent, is captured in the upcoming graphic, followed by the third formula.



Formula:

$$\psi_3 = \ominus\big((\neg b \wedge \neg\kappa)\mathcal{S}b\big) \wedge \neg\big((\neg b \wedge \neg\varphi_2)\mathcal{S}(\neg a \wedge \neg b \wedge \neg\varphi_2)\big) \wedge \neg b \wedge \bigcirc\neg(\varphi_1 \mathcal{U} \varphi_2)$$

By removing some unnecessary parts of the presented subformulas we observe the following elimination rule:

$$\varphi \equiv_i \varphi' = \psi_1 \vee \psi_2 \vee \psi_3$$

with

- $\psi_1 = b$
- $\psi_2 = \big((\neg b \wedge \neg\kappa)\,\mathcal{S}\,b\big) \wedge \bigcirc(\varphi_1\,\mathcal{U}\,\varphi_2)$
- $\psi_3 = \ominus\big((\neg b \wedge \neg\kappa)\,\mathcal{S}\,b\big) \wedge \neg\big((\neg b \wedge \neg\varphi_2)\,\mathcal{S}\,(\neg a \wedge \neg b \wedge \neg\varphi_2)\big) \wedge \neg b$
- $\kappa = \big((\neg b \wedge \neg\varphi_2)\,\mathcal{S}\,(\neg a \wedge \neg b \wedge \neg\varphi_2)\big) \wedge \varphi_1$

Elimination $3 - 14$

See Appendix A.

3.2 Processing on Formula-Level

After presenting Gabbay's Separation Theorem in detail, we now talk about its impact for our purposes. Giving the initially defined bounded synthesis problem

$$(I, O, i_0, \varphi)$$

with φ being a PLTL formula, we are able to apply Theorem 6 on φ to obtain a formula $\varphi' \equiv_i \varphi$. Thus, the solution of our initial synthesis problem is equivalent to the solution of the synthesis problem

$$(I, O, i_0, \varphi')$$

by the correctness of the separation theorem.

However, the theoretical correctness of this solution is irrelevant in practice, based on the non-elementary complexity of Theorem 6.

Chapter 4

The Allrounder: Using Tableau Construction

To present a solution for bounded synthesis that is also applicable in practice, we offer an approach using the tableau construction presented in Section 2.5. We assume the synthesis problem

$$(I, O, i_0, \varphi)$$

where input and output variables are disjunct and their union builds the set of atomic propositions, i.e., $AP = I \dot{\cup} O$. The formula φ is a PLTL formula over AP.

4.1 Creation of UCT

Based on the first step of original bounded synthesis, we are going to redefine Construction 5, allowing it to define a UCT out of a given PLTL formula φ. Our idea is very similar by observing the negation $\neg\varphi$ of our formula, building its NBW, and redefining it into a UCT. The creation of the NBW is completely based on the tableau construction in Section 2.5. Thus, we are left with the creation of a UCT describing $\mathcal{L}(\varphi)$. Its construction and proof are similar to the one presented in [16].

Construction 6. Given a PLTL formula φ, we define UCT $\mathcal{U}_\varphi$ using the following steps in between:

1) Build negation $\neg\varphi$ and transform it into NNF (Section 2.2.3).

2) Construction 2: $\neg\varphi$ to progressing 2VWAA $\mathcal{A}_{\neg\varphi}$.

3) Construction 3: $\mathcal{A}_{\neg\varphi}$ to transition-based GBW $\mathcal{G}_{\mathcal{A}_{\neg\varphi}}$

4) Construction 4: $\mathcal{G}_{\mathcal{A}_{\neg\varphi}}$ to NBW $\mathcal{N}_{\neg\varphi}$.

5) [16]: $\mathcal{N}_{\neg\varphi}$ to UCT $\mathcal{U}_\varphi$ simulating $\mathcal{N}_{\neg\varphi}$ along each path.

Thus, we finish with UCT $\mathcal{U}_\varphi$ such that $\mathcal{L}(\mathcal{U}_\varphi)$ describes $\mathcal{L}(\varphi)$.

Theorem 7. *Given a PLTL formula φ, we can construct a universal co-Büchi tree automaton $\mathcal{U}_\varphi$ with $2^{O(|\varphi|)}$ states that accepts a transition system $\mathcal{T}$ if, and only if, $\mathcal{T}$ satisfies φ.*

Proof. The correctness depends on the correctness of the constructions and the theorem presented in [16]. The number of states is determined by the exponential blowup of Construction 3, whereas all other constructions are linear in the size of formula φ.

□

Thus, given $\mathcal{U}_\varphi$, we are able to define a run graph on $\mathcal{T}$, following the definitions in Section 2.6.3.

Example. We apply our new approach for bounded synthesis on the PLTL formulas ξ_2 and ξ_3. At first, we build their negations and transform them into NNF (Section 2.2.3), i.e.,

$$\begin{aligned}\neg\xi_2 &= \Diamond(g_1 \wedge \boxminus\neg r_1) \wedge \Diamond(g_2 \wedge \boxminus\neg r_2)\\ &= \Big(\mathit{true}\,\mathcal{U}\,\big(g_1 \wedge (\mathit{false}\,\mathcal{T}\,\neg r_1)\big)\Big) \vee \Big(\mathit{true}\,\mathcal{U}\,\big(g_2 \wedge (\mathit{false}\,\mathcal{T}\,\neg r_2)\big)\Big),\end{aligned}$$

and

$$\begin{aligned}\neg\xi_3 &= \Diamond\big(g_1 \wedge \circledcirc(g_1\,\mathcal{T}\,\neg r_1)\big)\\ &= \mathit{true}\,\mathcal{U}\,\big(g_1 \wedge \circledcirc(g_1\,\mathcal{T}\,\neg r_1)\big).\end{aligned}$$

Now, we apply Construction 2 to both formulas. The related 2VWABAs are presented in Figure 4.1 and Figure 4.2, respectively. Applying Construction Construction 3 and Construction 4 results in the NBWs given in Figure 4.3 and Figure 4.4. These are equivalent to the final UCTs for ξ_2 and ξ_3. Finally, we build the run graphs of the UCTs on their respective transition systems, depicted in Figure 4.5 and Figure 4.6.

4.2 Constraint-based Approach

As we are able to define a UCT out of our given PLTL formula, the creation of a constraint system as input for an SMT solver is equivalent to the standard LTL approach, following the six steps presented in Section 2.6.5. Furthermore, also the size of the constrained system is computed the same, i.e., given UCT $\mathcal{U}_\varphi$ with transitions δ, the constraint system is of size

$$O(|\delta| \cdot |AP| + |I| \cdot |2^I|).$$

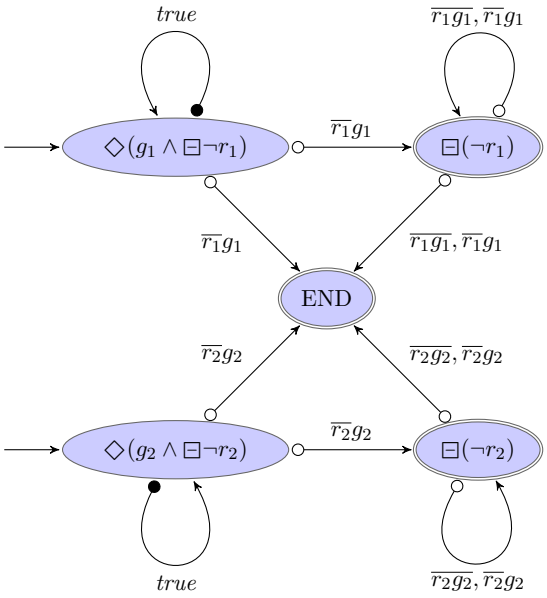


Figure 4.1: 2VWABA $\mathcal{A}_{\neg\xi_2}$ by applying Construction 2 on $\neg\xi_2$.

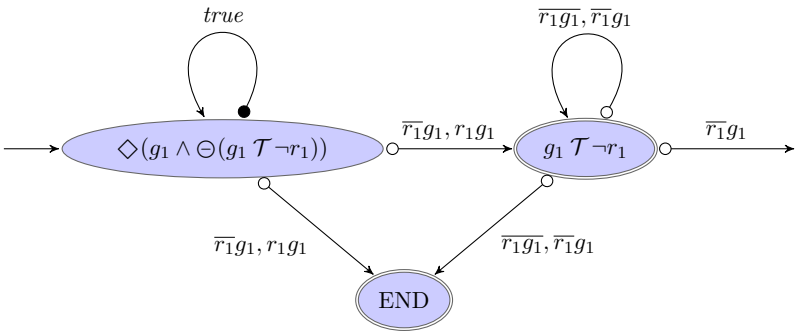


Figure 4.2: 2VWABA $\mathcal{A}_{\neg\xi_3}$ by applying Construction 2 on $\neg\xi_3$.

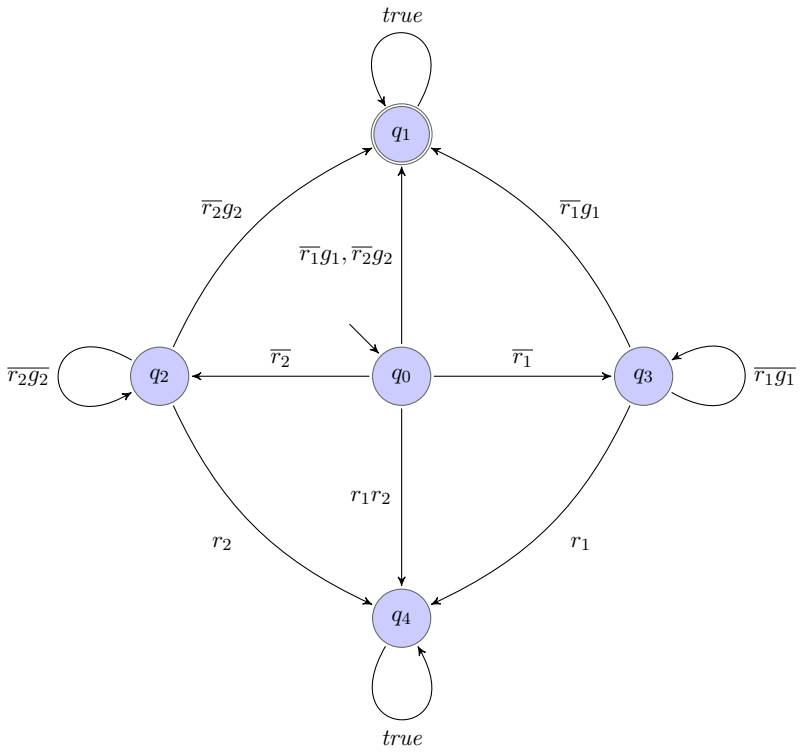


Figure 4.3: NBW $\mathcal{N}_{\neg\xi_2}$ that is equivalent to the result of Construction 3 and Construction 4 on $\mathcal{A}_{\neg\xi_2}$ in Figure 4.1. Furthermore, it is equivalent to the UCT $\mathcal{U}_{\xi_2}$.

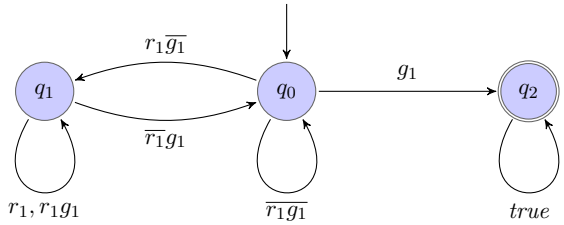


Figure 4.4: NBW $\mathcal{N}_{\neg\xi_3}$ that is equivalent to the result of Construction 3 and Construction 4 on $\mathcal{A}_{\neg\xi_3}$ in Figure 4.2. Furthermore, it is equivalent to the UCT $\mathcal{U}_{\xi_3}$.

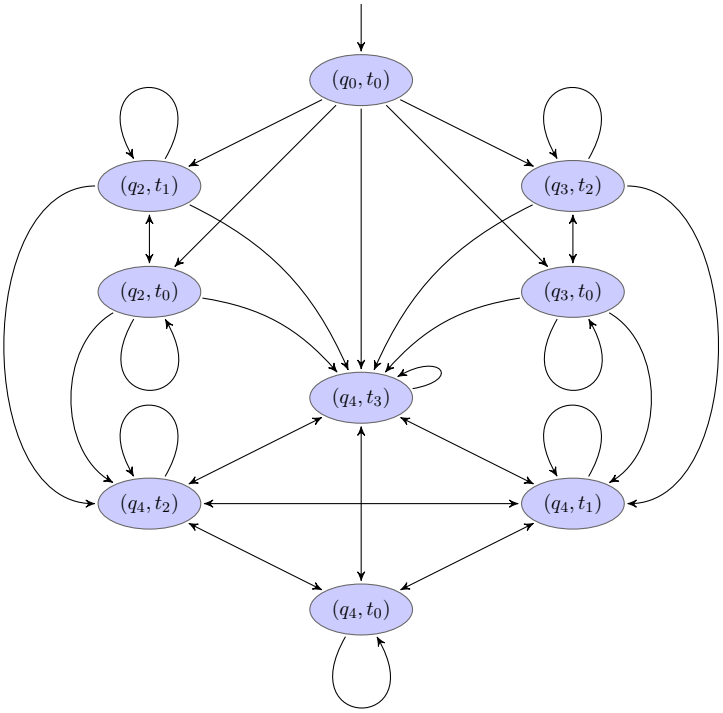


Figure 4.5: Run graph of $\mathcal{U}_{\xi_2}$ on $\mathcal{T}_{\xi_2}$. Since each of its vertices is annotated by 0, implementation $\mathcal{T}_{\xi_2}$ is valid.

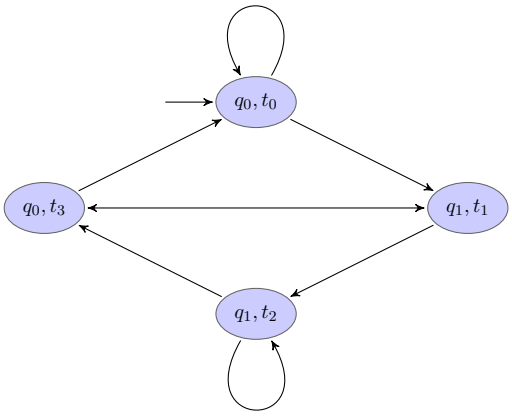


Figure 4.6: Run graph of $\mathcal{U}_{\xi_3}$ on $\mathcal{T}_{\xi_3}$. Since each of its vertices is annotated by 0, implementation $\mathcal{T}_{\xi_3}$ is valid.

Chapter 5

The Partial Approach: Exploiting Temporal Testers

The so-called *future(past)-fragment* of PLTL, where pure-past formulas are only connected by future and boolean operators, is a popular fragment of PLTL for describing specifications. Since it has been considered in many prior works [4, 7, 22], it encourages a special observation for bounded synthesis. We present a partial approach of bounded synthesis for this family of formulas using temporal testers. We assume the synthesis problem

$$(I, O, i_0, \varphi)$$

where input and output variables are disjunct and their union builds the set of atomic propositions, i.e., $AP = I \dot{\cup} O$. The formula φ is a PLTL formula over AP, built by n past formulas $\psi_1, \ldots, \psi_n$ that are separated by future and boolean connectives.

Before discussing our approach in detail, we present an equivalent and more intuitive representation for temporal testers using circuits. Afterwards, based on the original description of bounded synthesis, we start with the construction of a run graph, together with a valid annotation on it. Subsequently, we describe a solution for bounded synthesis using an SMT solver.

5.1 Representation of TTs as Circuits

Since temporal testers (Section 2.3.4) are fairness-free and deterministic systems, there is a more intuitive graphical representation of them using circuits. A circuit consists of input and output signals, as well as the inner gates AND, OR, NOT, and cells. As their names suggest, NOT-, AND-, and OR-gates are replacements for the related boolean operators. In addition, cells build a memory, storing a single boolean truth value. For gates and cells, we need a higher graphical representation:

NOT-gate AND-gate OR-gate Cell

In the following, we give a direct translation from PLTL to circuits. Afterwards, we show the equivalence between the circuit from Construction 7 and the temporal tester from Construction 1 for the same input formula φ.

Construction 7. Given a pure-past PLTL formula φ, a graphical representation for φ as circuit is obtained by the following translation:

- The inputs are defined by the vocabulary AP_φ.
- Cells are built by the set of all principally temporal subformulas of φ. For each such formula $\psi \in TSub(\varphi)$, we create an own cell, named x^c_ψ. Each such cell is initially *false*.
- The output is defined by the set of all principally temporal subformulas, so that, for each $\psi \in TSub(\varphi)$, we define an output x_ψ. Additionally, if φ has a boolean connective as outermost operator, we define an additional output x_φ.
- Further connections are build over the structure of φ using the graphical substitution in Figure 5.1 for some arbitrary formulas ψ_1 and ψ_2. Transparent black boxes require a further substitution of their inner formula. One may notice that this is similar to the definitions of χ and ρ of temporal testers.

  The substitution starts with the observation of output x_φ, connected to its blackbox including φ:

A computation π of a circuit is an infinite sequence of its outputs' truth values $X = \{x_\psi \mid \psi \in TSub(\varphi) \cup \{\varphi\}\}$ with $|X| = n$, i.e., $\pi = [X]_0[X]_1 \ldots \in (\mathbb{B}^n)^\omega$. To observe a specific output x_ψ with $\psi \in X$, we use the computation $\pi^\psi = [x_\psi]_0[x_\psi]_1 \ldots \in \mathbb{B}^\omega$.

Theorem 8. *For a pure-past PLTL formula φ, the graphical representation by circuit, obtained by Construction 7, is equivalent to the temporal tester $\mathcal{D}_\varphi = (V_\varphi, \theta_\varphi, \rho_\varphi)$ of Construction 1.*

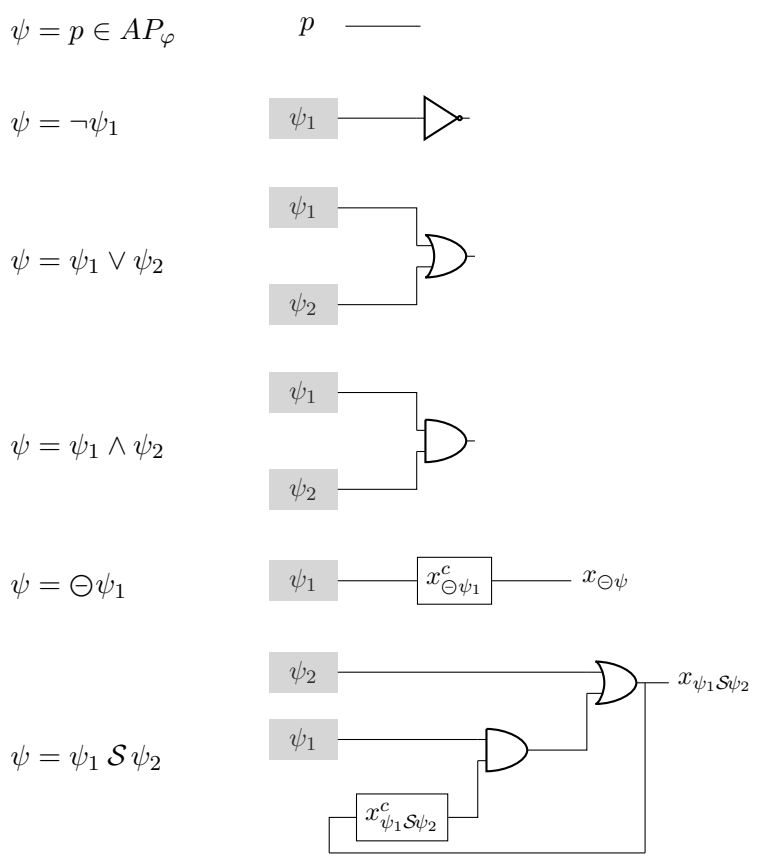


Figure 5.1: Substitution rules for the creation of a curcuit from a given pure-past PLTL formula ψ. Transparent black boxes require a further substitution of their inner formula.

Proof. Given a temporal tester $\mathcal{D}_\varphi = (V_\varphi, \theta_\varphi, \rho_\varphi)$ over AP, as well as a circuit for φ consisting of a set of inputs AP_φ, a set of gates, a set of cells X^c, and a set of outputs X, connected within the circuit, both created by Constructions 1 and 7, respectively. We show that $\mathcal{D}_\varphi$ and the circuit have the same behavior, i.e., given the sequence of inputs $\sigma = s_0 s_1 \ldots \in (2^{AP_\varphi})^\omega$, computations

- $\pi_1 = t_0 t_1 \ldots \in (2^{AP})^\omega$ of $\mathcal{D}_\varphi$ with $t_i|_{AP_\varphi} = s_i$, and
- $\pi_2 = [X]_0[X]_1 \ldots \in (\mathbb{B}^n)^\omega$ of the circuit

have the same truth values for any distinguished boolean variable x_ψ with $\psi \in TSub(\varphi) \cup \{\varphi\}$ in each time step. The proof is done by a structural induction over an arbitrary formula $\psi \in TSub(\varphi) \cup \{\varphi\}$, nested inside an induction over sequence σ.

For the *induction base*, we consider the initial input $s_0 \in 2^{AP_\varphi}$ of σ, implying states t_0 and $[X]_0$ of computations π_1 and π_2, respectively.

- For the *induction base* of the inner induction, we compare the truth values of $t_0[x_\psi]$ and $[x_\psi]_0 \in [X]_0$ for an arbitrary formula $\psi \in TSub(\varphi) \cup \{\varphi\}$. Given $p_1, p_2 \in AP_\varphi$, we make the following distinction:
  - If $\psi = p_1$, $\psi = \neg p_1$, $\psi = p_1 \vee p_2$, or $\psi = p_1 \wedge p_2$, then it depends in all cases directly on the inputs p_1, p_2, together with the boolean connective.
  - If $\psi = \ominus p_1$, then θ_φ determines $\neg x_\psi$, i.e., it is set to *false*. Output $x_{\ominus p_1}$ is given the value of its cell $x^c_{\ominus p_1}$, which is initialized with *false*. Thus, both are the same.
  - If $\psi = p_1 \mathcal{S} p_2$, then x_ψ is initialized in both cases by p_2.
- For the *induction hypothesis* of the inner induction, we assume that for all subformulas $\psi_1, \psi_2 \in TSub(\varphi) \cup \{\varphi\}$, the truth values of both computations are the same, i.e., $t_0[x_{\psi_1}] = [x_{\psi_1}]_0$ and $t_0[x_{\psi_2}] = [x_{\psi_2}]_0$.
- For its *induction step*, we observe an arbitrary formula $\psi \in TSub(\varphi) \cup \{\varphi\}$ that depends on at least one of both formulas of our induction hypothesis. We distinguish the following cases:
  - If $\psi = \neg\psi_1$, $\psi = \psi_1 \vee \psi_2$, or $\psi = \psi_1 \wedge \psi_2$, then both computations simply observe the boolean connectives between both truth values, which are given by the induction hypothesis.
  - If $\psi = \ominus\psi_1$, then, again, variable and output are set to *false*.
  - If $\psi = \psi_1 \mathcal{S} \psi_2$, then x_ψ is initialized in both cases by ψ_2, which is given by the induction hypothesis.

Thus, $t_0[x_\psi] = [x_\psi]_0$ for any formula $\psi \in TSub(\varphi) \cup \{\varphi\}$.

For the *induction hypothesis*, we assume that, up to position $n \in \mathbb{N}$ and for any temporal formula $\psi \in TSub(\varphi) \cup \{\varphi\}$, the prefixes of the computations π_1 and π_2 are the same for any variable x_ψ, i.e., $t_i[x_\psi] = [x_\psi]_i$ for $i \in \{0, \ldots, n\}$.

Finally, for the *induction step*, we observe the position $n+1$ of π_1 and π_2. Again, given the input $s_{n+1} \in 2^{AP_\varphi}$, we observe the states t_{n+1} and $[X]_{n+1}$ of computations π_1 and π_2, respectively.

- For the *induction base* of the inner induction, we compare the truth values of $t_{n+1}[x_\psi]$ and $[x_\psi]_{n+1}$ for an arbitrary formula $\psi \in TSub(\varphi) \cup \{\varphi\}$. Given $p_1, p_2 \in AP_\varphi$, we make the following distinction:

  - For any boolean formula over p_1 and p_2, the truth values only depend on the inputs and, thus, are the same.
  - If $\psi = \ominus p_1$, then, $t[x_\psi]_{n+1}$ is set to the prior value of x_{p_1} at time step n. Since, by construction, $[x_\psi]_{n+1}$ is given the same value by observing the truth value at its cell x^c_ψ, both are determined the same by the induction hypothesis.
  - If $\psi = p_1 \,\mathcal{S}\, p_2$, then, based on the definition of ρ, $t_{n+1}[x_\psi]$ is determined by $p_2' \vee \big(p_1' \wedge t_n[x_{\psi_1 \mathcal{S} \psi_2}]\big)$, where the primed versions consider the current input s_{n+1}. For the circuit, $[x_\psi]_{n+1}$ is defined in the same way, using the truth values of p_1 and p_2 at the current time step, and $[x_{\psi_1 \mathcal{S} \psi_2}]_n$. Since both are based on the expansion laws (Equation 2.1), the truth values are the same by the induction hypothesis.

- For the *induction hypothesis* of the inner induction, we assume that for all subformulas $\psi_1, \psi_2 \in TSub(\varphi) \cup \{\varphi\}$, the truth values of both computations at position $n+1$ are the same, i.e., $t_{n+1}[x_{\psi_1}] = [x_{\psi_1}]_{n+1}$ and $t_{n+1}[x_{\psi_2}] = [x_{\psi_2}]_{n+1}$.

- For its *induction step*, we observe some arbitrary subformula $\psi \in TSub(\varphi) \cup \{\varphi\}$ that depends on at least one of both formulas of our induction hypothesis. We distinguish the following cases:

  - Again, for any formula ψ with boolean connective as outermost operator, the truth values are the same, using the induction hypothesis.
  - If $\psi = \ominus \psi_1$, then, again, both variables are set to the prior value of ψ_1. Thus, by induction hypothesis, they are the same.
  - If $\psi = \psi_1 \,\mathcal{S}\, \psi_2$, then, by replacing the atomic propositions of the respective case during the induction base, the argumentation is similar using the induction hypothesis.

Thus, $t_{n+1}[x_\psi] = [x_\psi]_{n+1}$ for any formula $\psi \in TSub(\varphi) \cup \{\varphi\}$.

□

For simplicity, in the rest of this thesis, examples of temporal testers are given as circuits.

Example. Based on the final example of Section 2.3.4, we present the equivalent circuits for both TTs. Figure 5.2 shows the equivalent circuit of $\mathcal{D}_{\xi_2'}$. The equivalent representation of $\mathcal{D}_{\xi_3'}$ as a circuit is depicted in Figure 5.3.

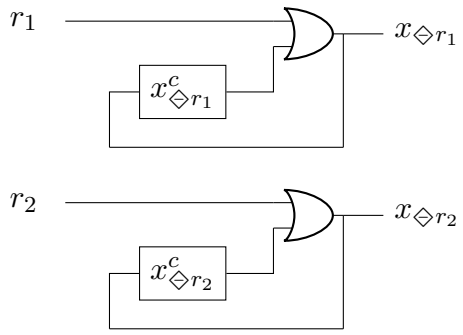


Figure 5.2: Temporal tester $\mathcal{D}_{\xi_2'}$ as circuit for the synchronous parallel composition of the inner formulas of ξ_2, i.e., $\xi_2^1 \mid\mid\mid \xi_2^2$. Both cells are initially empty.

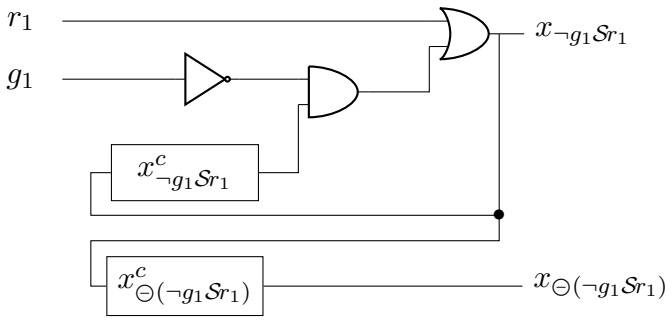


Figure 5.3: Temporal Tester $\mathcal{D}_{\xi_3'}$ as circuit for ξ_3^1. Both cells are initially empty.

5.2 Creation of Run Graph using TTs

Based on the original approach, we start with a guess of the implementation of formula φ, given as a 2^{AP}-labeled 2^I-transition system $\mathcal{T} = (T, t_0, \tau, o)$. In order to build a run graph of a universal co-Büchi tree automaton describing $\mathcal{L}(\varphi)$ on $\mathcal{T}$, we make a substitution of our given PLTL formula building a pure-future / LTL formula. For this, we replace in φ each of the pure-past subformulas $\psi_1, \ldots, \psi_n$ by an auxiliary and distinguishable boolean variable $x_{\psi_1}, \ldots, x_{\psi_n}$, resulting in formula φ'. Using Construction 5, we define a UCT $\mathcal{U}_{\varphi'}$ with

$$\mathcal{U}_{\varphi'} = \big(\Sigma, \Upsilon, Q, Q_0, \delta, \text{coBüchi}(C)\big)$$

where $\Sigma = 2^{AP \cup \{x_{\psi_1}, \ldots, x_{\psi_n}\}}$ and $\Upsilon = 2^I$. For each $\psi \in \{\psi_1, \ldots, \psi_n\}$, we define a fairness-free temporal tester $\mathcal{D}_\psi$ using Construction 1 such that

$$\mathcal{D}_\psi = (V_\psi, \theta_\psi, \rho_\psi).$$

By using the synchronous parallel composition $\mathcal{D}_{\psi_1} ||| \ldots ||| \mathcal{D}_{\psi_n}$ of all these temporal testers, we obtain a single temporal tester $\mathcal{D} = (V, \theta, \rho)$, testing for all truth values simultaneously. By X, we denote the set of x-variables or outputs of $\mathcal{D}$ ranging over a boolean domain by definition.

Given implementation $\mathcal{T}$, UCT $\mathcal{U}_{\varphi'}$, and temporal tester $\mathcal{D}$, we build the run graph $\mathcal{G} = (V, E)$ of $\mathcal{U}_{\varphi'}$ on $\mathcal{T}$ and $\mathcal{D}$, where the vertices are defined over triples with the last component describing an interpretation $\mathcal{X} \in 2^X$ over the variables of X, i.e., $V \subseteq Q \times T \times 2^X$, and the edges denote transitions over the vertices, i.e., $E \subseteq V \times V$. The run graph has to satisfy the following conditions:

- Each pair of initial states is a vertex in $\mathcal{G}$, i.e., $\forall\, q_0 \in Q_0.\, (q_0, t_0, \mathcal{X}_0) \in V$, where $\mathcal{X}_0 \vDash \theta[o(t_0)]$ describes the initial interpretation, further depending on initial input i_0.
- Each vertex $(q, t, \mathcal{X}) \in V$ satisfies the underlying transition relations, i.e.,

$$\forall\, (q, t, \mathcal{X}) \in V. \Big\{(q', \upsilon) \in Q \times \Upsilon \mid \Big(\big(q, t, \mathcal{X}\big), \big(q', \tau(t, \upsilon), \mathcal{X}'\big)\Big) \in E\Big\}$$
$$\text{satisfies } \delta\big(q, o(t) \cup \mathcal{X}\big)$$

  with

$$(\mathcal{X}, \mathcal{X}') \vDash \rho\big[o\big(\tau(t, \upsilon)\big)\big].$$

After this, we build an annotation $\lambda : V \to \{\lrcorner\} \cup \mathbb{N}$ on $\mathcal{G}$, indicating the maximal number of rejecting states occurring on some path to a node in the run graph. Its denotations are similar to those for normal bounded synthesis. We call λ valid if it satisfied the following conditions:

- The initial vertices are annotated by a natural number, formally, $\forall\, q_0 \in Q_0.\, \lambda(q_0, t_0, \mathcal{X}_0) \neq \lrcorner$.
- Each path of $\mathcal{G}$ has an increasing annotation, which is strictly increasing whenever another rejecting state is encountered, formally, for some vertex $(q, t, \mathcal{X}) \in V$ with $(q', \upsilon) \in \delta\big(q, o(t) \cup \mathcal{X}\big)$ and $(\mathcal{X}, \mathcal{X}') \vDash \rho\big[o\big(\tau(t, \upsilon)\big)\big]$:

$$\lambda(q, t, \mathcal{X}) \begin{cases} < \lambda\big(q', \tau(t, \upsilon), \mathcal{X}'\big), & \text{if } q' \in \mathcal{C} \\ \leq \lambda\big(q', \tau(t, \upsilon), \mathcal{X}'\big), & \text{otherwise.} \end{cases}$$

Theorem 9. *A finite-state Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is accepted by a universal co-Büchi tree automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, q_0, \delta,$ coBüchi$(C))$ together with a temporal tester $\mathcal{D} = (V, \theta, \rho)$ if, and only if, it has a valid $(|T| \cdot |C| \cdot 2^{|X|})$-bounded annotation.*

Proof. Since $\mathcal{U}$ is universal and $\mathcal{D}$ is fairness-free and deterministic with respect to any sequence of inputs, $\mathcal{U}$ and $\mathcal{D}$ have a unique run graph $\mathcal{G} = (G, E)$. Since $\mathcal{T}$, $\mathcal{U}$, and $\mathcal{D}$ are finite, $\mathcal{G}$ is finite, too.

If $\mathcal{G}$ contains a lasso with a rejecting state in its loop, i.e., paths

$$(q_0, t_0, \mathcal{X}_0)(q_1, t_1, \mathcal{X}_1) \dots (q_k, t_k, \mathcal{X}_k)$$

and

$$(q'_0, t'_0, \mathcal{X}'_0)(q'_1, t'_1, \mathcal{X}'_1) \dots (q'_l, t'_l, \mathcal{X}'_l)$$

such that

$$(q_k, t_k, \mathcal{X}_k) = (q'_0, t'_0, \mathcal{X}'_0) = (q'_l, t'_l, \mathcal{X}'_l)$$

and

$$(q'_i, t'_i, \mathcal{X}'_i) \text{ is rejecting for some } i \in \{1, \dots, l\},$$

then, by induction, any valid annotation λ satisfies

- $\lambda(q_j, t_j, \mathcal{X}_j) \in \mathbb{N}$ for all $j \in \{0, \dots, k\}$,
- $\lambda(q'_j, t'_j, \mathcal{X}'_j) \in \mathbb{N}$ for all $j \in \{0, \dots, l\}$,
- $\lambda(q'_{j-1}, t'_{j-1}, \mathcal{X}'_{j-1}) \leq \lambda(q'_j, t'_j, \mathcal{X}'_j)$ for all $j \in \{0, \dots, l\}$,
- $\lambda(q'_{i-1}, t'_{i-1}, \mathcal{X}'_{i-1}) < \lambda(q'_i, t'_i, \mathcal{X}'_i)$.

This contradicts that $\mathcal{G}$ is finite. $\lightning$

If, on the other hand, $\mathcal{G}$ does not contain a lasso with a rejecting state in its loop, we can infer a valid $(|T| \cdot |C| \cdot 2^{|X|})$-bounded annotation by assigning to each vertex $(q, t, \mathcal{X}) \in V$ of $\mathcal{G}$ the highest number of rejecting states occuring on some path $(q_0, t_0, \mathcal{X}_0)(q_1, t_1, \mathcal{X}_1) \dots (q_n, t_n, \mathcal{X}_n)$, and by assigning $\lrcorner$ to every pair of states $(q, t, \mathcal{X}) \notin V$.

□

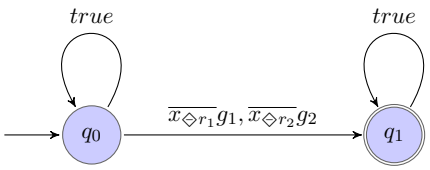


Figure 5.4: UCT $\mathcal{U}_{\xi_2'}$ for ξ_2'.

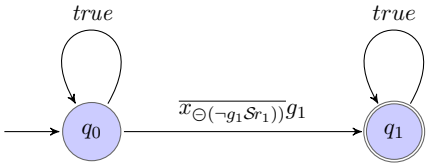


Figure 5.5: UCT $\mathcal{U}_{\xi_3'}$ for ξ_3'.

Example. We present the run graphs for the future(past)-formulas ξ_2 and ξ_3. Parts of their components were already defined in previous examples: implementation $\mathcal{T}_{\xi_2}$ in Figure 2.3, and TT $\mathcal{D}_{\xi_2'}$ in Figure 5.2, as well implementation $\mathcal{T}_{\xi_3}$ in Figure 2.4, and TT $\mathcal{D}_{\xi_3'}$ in Figure 5.3.

Figure 5.4 shows the UCT $\mathcal{U}_{\xi_2'}$ for the substitution formula $\xi_2' = \square(g_1 \rightarrow x_{\Diamond r_1}) \wedge \square(g_2 \rightarrow x_{\Diamond r_2})$, Figure 5.5 the UCT $\mathcal{U}_{\xi_3'}$ with $\xi_3' = \square(g_1 \rightarrow x_{\Diamond(\neg g_1 \mathcal{S} r_1)})$.

Finally, Figure 5.6 gives the run graph of $\mathcal{U}_{\xi_2'}$ on $\mathcal{T}_{\xi_2}$ and $\mathcal{D}_{\xi_2'}$, Figure 5.6 the run graph of $\mathcal{U}_{\xi_3'}$ on $\mathcal{T}_{\xi_3}$ and $\mathcal{D}_{\xi_3'}$.

5.3 Constraint-based Adaption

For the constraint-based approach using an SMT solver, we have to make some adaptions. However, based on the symbolic representation of temporal testers, we can reuse its inner formulas, which simplifies the upcoming formulation.

To construct our constraint system, we assume the existence of a temporal tester $\mathcal{D} = (V, \theta, \rho)$ with set of distinguishable boolean variables X for the inner pure-past formulas of input formula φ over AP, as well as a UCT $\mathcal{U}_{\varphi'} = (\Sigma, \Upsilon, Q, Q_0, \delta, \text{coBüchi}(C))$ describing substitution formula φ'. We want to specify the existence of a finite input-preserving 2^{AP}-labeled 2^I-transition system $\mathcal{T} = (T, t_0, \tau, o)$ that is accepted by $\mathcal{U}_{\varphi'}$ in combination with $\mathcal{D}$ and has a valid annotation λ. This is done in multiple steps:

1. *Encode transition function τ*

   We introduce for each direction $\upsilon \in 2^I$ a unary function symbol $\tau_\upsilon : T \rightarrow T$ that maps each state $t \in T$ to its successor on direction υ, formally, $\tau_\upsilon(t) = \tau(t, \upsilon)$.

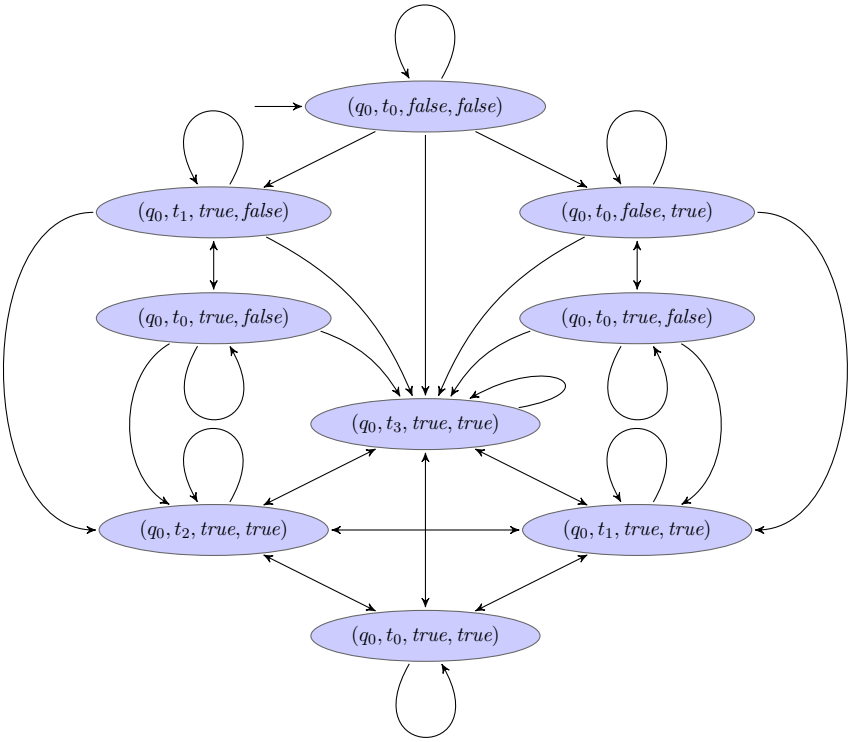


Figure 5.6: Run graph of $\mathcal{U}_{\xi_2'}$ on $\mathcal{T}_{\xi_2}$ and $\mathcal{D}_{\xi_2'}$. Based on our implementation, each of its vertices is annotated by 0, validating implementation $\mathcal{T}_{\xi_2}$.

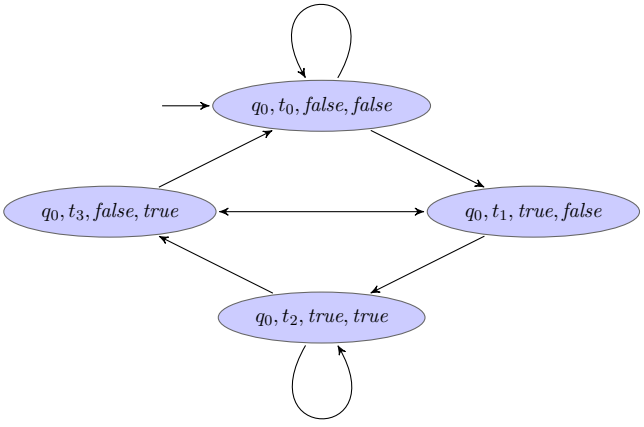


Figure 5.7: Run graph of $\mathcal{U}_{\xi_3'}$ on $\mathcal{T}_{\xi_3}$ and $\mathcal{D}_{\xi_3'}$. Based on our implementation, each of its vertices is annotated by 0, validating implementation $\mathcal{T}_{\xi_3}$.

2. *Encode labeling function o*

   In order to decide if a state $t \in T$ is labeled with proposition $a \in AP$, we introduce for each proposition a unary predicate symbol $a : T \to \mathbb{B}$. It maps a state to *true* if its underlying proposition is part of the state, i.e., $a(t) = \mathit{true}$ iff $a \in o(t)$.

3. *Encode temporal tester $\mathcal{D}$*

   We introduce for each variable $x_\psi \in X$, i.e., for each subformula ψ, a unary predicate symbol $x_\psi : T \to \mathbb{B}$ that indicates if variable x_ψ is assigned to *true*.

   For each state $t \in T$ and direction $\upsilon \in 2^I$, we encode the conjunction ρ by transforming each assignment $x'_\psi \leftrightarrow \mu$ into a constraint $x_\psi\big(\tau_\upsilon(t)\big) \leftrightarrow \mu'$, such that each occurring x_ψ in μ is replaced by $x_\psi(t)$ in μ'.

4. *Encode annotation λ*

   For our annotation, we introduce for each state $q \in Q$ of UCT $\mathcal{U}_\varphi$ two unary symbols.

   - $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$ where it maps state $t \in T$ to *true* if, and only if, its annotation is a natural number, formally, $\lambda_q^{\mathbb{B}}(t) = \mathit{true}$ iff $\lambda(q, t, \mathcal{X}) \in \mathbb{N}$.
   - $\lambda_q^{\#} : T \to \mathbb{N}$ where it maps state $t \in T$ to its annotation value $\lambda(q, t, \mathcal{X})$, or is undefined if $\lambda(q, t, \mathcal{X}) = \text{␣}$, formally,

   $$\lambda_q^{\#}(t) = \begin{cases} \lambda(q, t, \mathcal{X}) & \text{if } \lambda(q, t, \mathcal{X}) \in \mathbb{N} \\ \text{undef.} & \text{if } \lambda(q, t, \mathcal{X}) = \text{␣}. \end{cases}$$

   Note that each interpretation of variables $\mathcal{X}$ directly depends on the labeling of underlying state $t \in T$, and, thus, can be ignored for the annotation function.

5. *Formalization of valid annotation for $\mathcal{T}$*

   We first define the following abbreviations:

   - For some state $t \in T$, the symbol $\overrightarrow{a} : T \to 2^{AP}$ represents the labeling of t, formally, $\overrightarrow{a}(t) = o(t)$.
   - For some state $t \in T$, the symbol $\overrightarrow{x} : T \to 2^X$ represents interpretation $\mathcal{X}$ of state t.
   - To update the interpretation for some given state $t \in T$ and direction $\upsilon \in 2^I$, we introduce the symbol $\overrightarrow{x}_t^{\upsilon}$. It represents a conjunction that updates for each $x_\psi \in X$ each truth value $x_\psi\big(\tau_\upsilon(t)\big)$. It is build by substitution on conjunction ρ with $x_\psi \in X$ and $a \in AP$:

  - Each occurrence of x'_ψ is replaced by $x_\psi\big(\tau_\upsilon(t)\big)$,
  - Each occurrence of x_ψ is replaced by $x_\psi(t)$,
  - Each occurrence of a' is replaced by $a\big(\tau_\upsilon(t)\big)$, and
  - Each occurrence of a is replaced by $a(t)$.
- For some state $q \in Q$, the symbol $\rhd_q$ gives the correct greater sign, i.e., $\rhd_q$ is $>$ if $q \in F$, and $\geq$ otherwise.

Thus, in order to achieve a valid annotation of $\mathcal{T}$, we formalize the constraint

$$\begin{aligned}\forall\, t.\, \lambda_q^{\mathbb{B}}(t) \wedge (q', \upsilon) \in \delta\big(q, \overrightarrow{a}(t) \cup \overrightarrow{x}(t)\big) \\ \rightarrow \overrightarrow{x}_t^{\upsilon} \wedge \lambda_{q'}^{\mathbb{B}}\big(\tau_\upsilon(t)\big) \wedge \lambda_{q'}^{\#}\big(\tau_\upsilon(t)\big) \rhd_{q'} \lambda_q^{\#}(t).\end{aligned}$$

Additionally, we require that all pairs of initial states are initially labeled by a natural number (w.l.o.g. 0), formally, $\forall\, q_0 \in Q_0.\, \lambda_{q_0}^{\mathbb{B}}(t_0)$.

6. *Guarantee input-preservation of $\mathcal{T}$*

   By the definition of input-preserving, we have to guarantee that the label of each state $t \in T$ reflects the previous input. Therefore, for each input symbol $a \in I$ and each direction $\upsilon \in 2^I$, we have to specify whether $a \in \upsilon$. We add the constraints

   - $\forall\, t.\, a\big(\tau_\upsilon(t)\big)$ if $a \in \upsilon$, and
   - $\forall\, t.\, \neg a\big(\tau_\upsilon(t)\big)$ if $a \notin \upsilon$.

7. *Initial labeling of t_0 and $\mathcal{X}_0$*

   The initial state t_0 is labeled with the initial input i_0, i.e., we build the conjunction $p_1(t_0) \wedge \ldots \wedge p_n(t_0)$ with $i_0 = \{p_1, \ldots, p_n\}$.

   Furthermore, the initial interpretation is determined by the initial state t_0, which further depends on i_0. Thus, we make another substitution on θ with $x_\psi \in X$ and $a \in AP$:

   - Each occurrence of x_ψ is replaced by $x_\psi(t)$, and
   - Each occurrence of a is replaced by $a(t)$.

Such a constraint system, resulting from a given future(past) PLTL specification, is satisfiable (modulo a theory with order) if, and only if, the PLTL specification is finite-state realizable. Furthermore, given UCT $\mathcal{U}_{\varphi'}$ for LTL formula φ', as well as temporal tester $\mathcal{D}_\varphi$ for its inner pure-past formulas with set of variables X, the constraint system has size

$$O\big(2^{|X|} \cdot (|\delta| \cdot |AP| + |I| \cdot 2^{|I|})\big).$$

By completely resolving the universal quantification of $\mathcal{U}_{\varphi'}$, the size of the constraint system additionally depends on size $b_{\mathcal{T}}$ of the constraint system $\mathcal{T}$, leading to the formula

$$O\big(b_{\mathcal{T}} \cdot 2^{|X|} \cdot (|\delta| \cdot |AP| + |I| \cdot 2^{|I|})\big).$$

Example. We are now able to define the constraints for example formulas ξ_2 and ξ_3 as inputs for an SMT solver. For readability reasons, we copied the symbolic representations for the temporal testers $\mathcal{D}_{\xi_2'} = (V_{\xi_2'}, \theta_{\xi_2'}, \rho_{\xi_2'})$ and $\mathcal{D}_{\xi_3'} = (V_{\xi_3'}, \theta_{\xi_3'}, \rho_{\xi_3'})$ with

- $V_{\xi_2'} = \{r_1, r_2\} \cup \{x_{\Diamond r_1}, x_{\Diamond r_2}\}$
- $\theta_{\xi_2'} := (x_{\Diamond r_1} \leftrightarrow r_1) \wedge (x_{\Diamond r_2} \leftrightarrow r_2)$
- $\rho_{\xi_2'} := (x'_{\Diamond r_1} \leftrightarrow r_1' \vee x_{\Diamond r_1}) \wedge (x'_{\Diamond r_2} \leftrightarrow r_2' \vee x_{\Diamond r_2})$

and

- $V_{\xi_2'} = \{r_1, g_1\} \cup \{x_{\neg g_1 \mathcal{S} r_1}, x_{\ominus(\neg g_1 \mathcal{S} r_1)}\}$
- $\theta_{\xi_2'} := \neg x_{\ominus(\neg g_1 \mathcal{S} r_1)} \wedge (x_{\neg g_1 \mathcal{S} r_1} \leftrightarrow r_1)$
- $\rho_{\xi_2'} := \big(x'_{\neg g_1 \mathcal{S} r_1} \leftrightarrow r_1' \vee (g_1' \wedge x_{\neg g_1 \mathcal{S} r_1})\big) \wedge \big(x'_{\ominus(\neg g_1 \mathcal{S} r_1)} \leftrightarrow x_{\neg g_1 \mathcal{S} r_1}\big)$

such that $X_{\xi_2} = \{x_{\Diamond r_1}, x_{\Diamond r_2}\} \subset V_{\xi_2}$ and $X_{\xi_3} = \{x_{\ominus(\neg g_1 \mathcal{S} r_1))}\} \subset V_{\xi_3}$.

Furthermore, we assume the UCT $\mathcal{U}_{\xi_2'}$ in Figure 5.4, as well as UCT $\mathcal{U}_{\xi_3'}$ in Figure 5.5.

Starting with ξ_2, we specify the existence of a finite input-preserving $2^{\{r_1,r_2,g_1,g_2\}}$-labeled $2^{\{r_1,r_2\}}$-transition system $\mathcal{T}_{\xi_2} = (T, t_0, \tau, o)$:

1. *Encode transition function τ*

   Unary function symbol $\tau_\upsilon : T \to T$ for each direction $\upsilon \in 2^{\{r_1,r_2\}}$.

2. *Encode labeling function o*

   Unary predicate symbol $a : T \to \mathbb{B}$ for each proposition $a \in \{r_1, r_2, g_1, g_2\}$.

3. *Encode temporal tester $\mathcal{D}$*

   Unary predicate symbol $x_\psi : T \to \mathbb{B}$ for each variable $x_\psi \in \{x_{\Diamond r_1}, x_{\Diamond r_2}\}$.

4. *Encode annotation λ*

   Unary symbols $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$ and $\lambda_q^{\#} : T \to \mathbb{N}$ for each state $q \in Q$.

5. *Formalization of valid annotation for $\mathcal{T}$*

   We define symbol $\overrightarrow{x}_t^{\upsilon}$ by applying the substitutions:

$$\overrightarrow{x}_t^{\upsilon} = \Big(x_{\Diamond r_1}\big(\tau_\upsilon(t)\big) \leftrightarrow r_1\big(\tau_\upsilon(t)\big) \vee x_{\Diamond r_1}(t)\Big) \\ \wedge \Big(x_{\Diamond r_2}\big(\tau_\upsilon(t)\big) \leftrightarrow r_2\big(\tau_\upsilon(t)\big) \vee x_{\Diamond r_2}(t)\Big).$$

   Now, we encode each transition of $\mathcal{U}_{\xi_2'}$ separately. (a) and (c) encode the self-loops for q_0 and q_1, respectively, and (b) the remaining transition.

$$\begin{aligned}
\text{a)}\ \forall t.\, \lambda_0^{\mathbb{B}}(t) \ \rightarrow\ & \overrightarrow{x}_t^{\overline{r_1 r_2}} \wedge \lambda_0^{\mathbb{B}}\big(\tau_{\overline{r_1 r_2}}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{\overline{r_1 r_2}}(t)\big) \geq \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{\overline{r_1} r_2} \wedge \lambda_0^{\mathbb{B}}\big(\tau_{\overline{r_1} r_2}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{\overline{r_1} r_2}(t)\big) \geq \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 \overline{r_2}} \wedge \lambda_0^{\mathbb{B}}\big(\tau_{r_1 \overline{r_2}}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{r_1 \overline{r_2}}(t)\big) \geq \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 r_2} \wedge \lambda_0^{\mathbb{B}}\big(\tau_{r_1 r_2}(t)\big) \wedge \lambda_0^{\#}\big(\tau_{r_1 r_2}(t)\big) \geq \lambda_0^{\#}(t) \\
\text{b)}\ \forall t.\, \lambda_0^{\mathbb{B}}(t) \ \wedge\ & \Big(\big(\neg x_{\Diamond r_1}(t) \wedge \neg r_1(t) \wedge g_1(t)\big) \\
& \vee\big(\neg x_{\Diamond r_2}(t) \wedge \neg r_2(t) \wedge g_2(t)\big)\Big) \\
\rightarrow\ & \overrightarrow{x}_t^{\overline{r_1 r_2}} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1 r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1 r_2}}(t)\big) > \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{\overline{r_1} r_2} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1} r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1} r_2}(t)\big) > \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 \overline{r_2}} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{r_1 \overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1 \overline{r_2}}(t)\big) > \lambda_0^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 r_2} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{r_1 r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1 r_2}(t)\big) > \lambda_0^{\#}(t) \\
\text{c)}\ \forall t.\, \lambda_1^{\mathbb{B}}(t) \ \rightarrow\ & \overrightarrow{x}_t^{\overline{r_1 r_2}} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1 r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1 r_2}}(t)\big) > \lambda_1^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{\overline{r_1} r_2} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{\overline{r_1} r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{\overline{r_1} r_2}(t)\big) > \lambda_1^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 \overline{r_2}} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{r_1 \overline{r_2}}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1 \overline{r_2}}(t)\big) > \lambda_1^{\#}(t) \\
\wedge\ & \overrightarrow{x}_t^{r_1 r_2} \wedge \lambda_1^{\mathbb{B}}\big(\tau_{r_1 r_2}(t)\big) \wedge \lambda_1^{\#}\big(\tau_{r_1 r_2}(t)\big) > \lambda_1^{\#}(t)
\end{aligned}$$

   Furthermore, all pairs of initial states are labeled with 0, i.e., $\lambda_0^{\mathbb{B}}(0)$.

6. *Guarantee input-preservation of $\mathcal{T}$*

$$\forall t.\, r_1(\tau_{r_1 r_2}(t)) \wedge r_2(\tau_{r_1 r_2}(t)) \wedge r_1(\tau_{r_1 \overline{r_2}}(t)) \wedge \neg r_2(\tau_{r_1 \overline{r_2}}(t)) \\ \neg r_1(\tau_{\overline{r_1} r_2}(t)) \wedge r_2(\tau_{\overline{r_1} r_2}(t)) \wedge \neg r_1(\tau_{\overline{r_1 r_2}}(t)) \wedge \neg r_2(\tau_{\overline{r_1 r_2}}(t))$$

7. *Initial labeling of t_0 and $\mathcal{X}_0$*

$$\neg r_1(0) \wedge \neg r_2(0) \wedge \big(x_{\Diamond r_1}(0) \leftrightarrow r_1(0)\big) \wedge \big(x_{\Diamond r_2}(0) \leftrightarrow r_1(0)\big),$$

   which can be directly reduced to

$$\neg r_1(0) \wedge \neg r_2(0) \wedge \neg x_{\Diamond r_1}(0) \wedge \neg x_{\Diamond r_2}(0).$$

Subsequently, we now define the constraints for formula ξ_3. We specify the existence of a finite input-preserving $2^{\{r_1, g_1\}}$-labeled $2^{\{r_1\}}$-transition system $\mathcal{T}_{\xi_3} = (T, t_0, \tau, o)$:

1. *Encode transition function τ*

   Unary function symbol $\tau_\upsilon : T \to T$ for each direction $\upsilon \in 2^{\{r_1\}}$.

2. *Encode labeling function o*

   Unary predicate symbol $a : T \to \mathbb{B}$ for each proposition $a \in \{r_1, g_1\}$.

3. *Encode temporal tester $\mathcal{D}$*

   Unary predicate symbol $x_\psi : T \to \mathbb{B}$ for each variable $x_\psi \in \{x_{\neg g_1 \mathcal{S} r_1}, x_{\ominus(\neg g_1 \mathcal{S} r_1)}\}$. For readability, the set is abbreviated by $\{x_\mathcal{S}, x_\ominus\}$.

4. *Encode annotation λ*

   Unary symbols $\lambda_q^\mathbb{B} : T \to \mathbb{B}$ and $\lambda_q^\# : T \to \mathbb{N}$ for each state $q \in Q$.

5. *Formalization of valid annotation for $\mathcal{T}$*

   Again, we define symbol $\overrightarrow{x}_t^\upsilon$ by applying the substitutions:

$$\begin{aligned}\overrightarrow{x}_t^\upsilon = &\Big(x_\mathcal{S}\big(\tau_\upsilon(t)\big) \leftrightarrow r_1\big(\tau_\upsilon(t)\big) \vee \Big(g_1\big(\tau_\upsilon(t)\big) \wedge x_\mathcal{S}(t)\Big)\Big) \\ &\wedge \Big(x_\ominus\big(\tau_\upsilon(t)\big) \leftrightarrow x_\mathcal{S}(t)\Big).\end{aligned}$$

   We encode the transitions of $\mathcal{U}_{\xi_3'}$ where (a) and (c) describe the self-loops of q_0 and q_1, respectively, and (b) the remaining transition.

$$\begin{aligned}
\text{a)}\ \forall t.\, \lambda_0^\mathbb{B}(t) \ &\to\ \overrightarrow{x}_t^{\overline{r_1}} \wedge \lambda_0^\mathbb{B}\big(\tau_{\overline{r_1}}(t)\big) \wedge \lambda_0^\#\big(\tau_{\overline{r_1}}(t)\big) \geq \lambda_0^\#(t) \\
&\wedge\ \overrightarrow{x}_t^{r_1} \wedge \lambda_0^\mathbb{B}\big(\tau_{r_1}(t)\big) \wedge \lambda_0^\#\big(\tau_{r_1}(t)\big) \geq \lambda_0^\#(t) \\
\text{b)}\ \forall t.\, \lambda_0^\mathbb{B}(t) \ &\wedge\ \neg x_\mathcal{S}(t) \wedge g_1(t) \\
&\to\ \overrightarrow{x}_t^{\overline{r_1}} \wedge \lambda_1^\mathbb{B}\big(\tau_{\overline{r_1}}(t)\big) \wedge \lambda_1^\#\big(\tau_{\overline{r_1}}(t)\big) > \lambda_1^\#(t) \\
&\wedge\ \overrightarrow{x}_t^{r_1} \wedge \lambda_1^\mathbb{B}\big(\tau_{r_1}(t)\big) \wedge \lambda_1^\#\big(\tau_{r_1}(t)\big) > \lambda_1^\#(t) \\
\text{c)}\ \forall t.\, \lambda_1^\mathbb{B}(t) \ &\to\ \overrightarrow{x}_t^{\overline{r_1}} \wedge \lambda_1^\mathbb{B}\big(\tau_{\overline{r_1}}(t)\big) \wedge \lambda_1^\#\big(\tau_{\overline{r_1}}(t)\big) > \lambda_1^\#(t) \\
&\wedge\ \overrightarrow{x}_t^{r_1} \wedge \lambda_1^\mathbb{B}\big(\tau_{r_1}(t)\big) \wedge \lambda_1^\#\big(\tau_{r_1}(t)\big) > \lambda_1^\#(t)
\end{aligned}$$

   For the initial labeling, we have $\lambda_0^\mathbb{B}(0)$.

6. *Guarantee input-preservation of $\mathcal{T}$*

$$\forall t.\, r_1\big(\tau_{r_1}(t)\big) \wedge \neg r_1\big(\tau_{\overline{r_1}}(t)\big).$$

7. *Initial labeling of t_0 and $\mathcal{X}_0$*

$$\neg r_1(0) \wedge \neg x_\ominus(0) \wedge \neg x_\mathcal{S}(0).$$

Chapter 6

Comparison between the Approaches

At the time of writing the thesis, it presents the first approach for bounded synthesis for PLTL. Thus, we are unable to compare our algorithms to other approaches. However, we are able to compare our three derived approaches. We are interested in finding advantages and drawbacks of our approaches that make them applicable in different situations or dedicated settings. Because of its non-elementary complexity, we quit to include the naive approach from our comparison. Thus, we make a comparison between allrounder (Chapter 4) and partial approach (Chapter 5). First, we present a family of specifications depending on some factor $n \in \mathbb{N}^+$ in the future(past)-fragment that is hard to deal within the partial approach, whereas its observation within the allrounder is more efficient referring the sizes of the constraint systems. Afterwards, we list the partial approach's main advantages, justifying its consideration and further observation. To further motivate the latter, we present an idea for a fourth approach that forms an approximative solution.

6.1 Advantage of the Allrounder

Given some number $n \in \mathbb{N}^+$, we assume $\log(n)$ input signals $r_1, \ldots, r_{\log(n)}$ and $\log(n)$ output signals $g_1, \ldots, g_{\log(n)}$. We encode their indices as bits, which leads to $2^{\log(n)} = n$ different combinations of input and output signals, in the following defined by $R_1, \ldots, R_n$ and $G_1, \ldots, G_n$, respectively. Our family of formulas states that if a combination of output signals is given, then the respective combination of input signals was observed in the past.

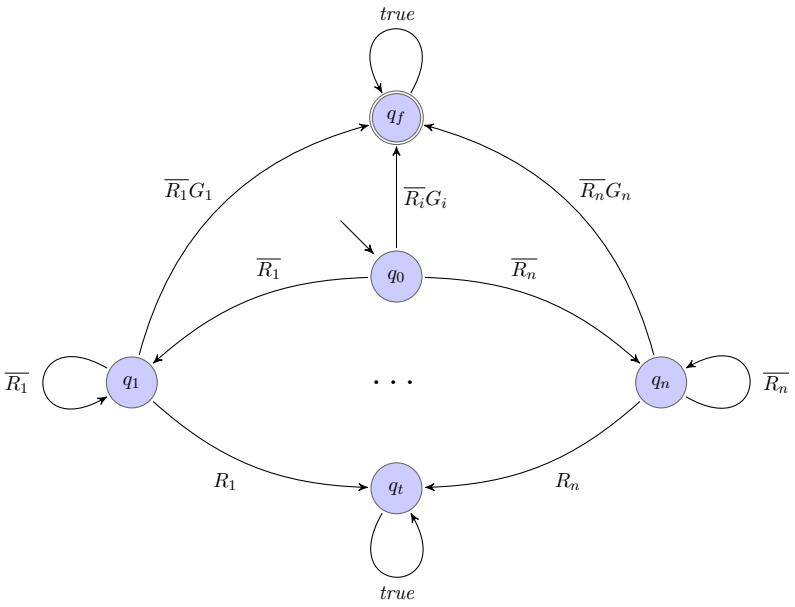


Figure 6.1: UCT $\mathcal{U}_\varphi$ for formula φ. Its idea is that each request combination that was not sent at the first step is observed separately. Whenever a grant combination is given without having seen its respective request combination before, the formula is violated.

Formally:

$$\begin{aligned}\{\varphi \;=\; & \square(\overline{g_n \dots g_1} \rightarrow \diamondminus \overline{r_n \dots r_1}) \\ \wedge\; & \square(\overline{g_n \dots g_2} g_1 \rightarrow \diamondminus \overline{r_n \dots r_2} r_1) \\ \wedge\; & \square(\overline{g_n \dots g_3} g_2 \overline{g_1} \rightarrow \diamondminus \overline{r_n \dots r_3} r_2 \overline{r_1}) \\ & \dots \\ \wedge\; & \square(g_n \dots g_1 \rightarrow \diamondminus r_n \dots r_1) \mid n \in \mathbb{N}\}\end{aligned}$$

We apply both approaches on the synthesis problem

$$(I, O, i_0, \varphi)$$

where input and output variables are disjunct and their union builds the set of atomic propositions, i.e., $AP = I \dot\cup O$.

First, we observe the formula φ for the allrounder approach. Its UCT $\mathcal{U}_\varphi$ is shown in Figure 6.1. The sizes for $\mathcal{U} = \big(\Sigma, \Upsilon, Q, \{q_0\}, \delta, \text{coBüchi}(C)\big)$ are given by:

- $|Q| = n + 3$

- $|\delta| \in O(n^3)$
- $|C| = 1$

For the partial approach, we start with a substitution for each past-formula $\diamondsuit R_i$ by distinguishable variables $x_{\diamondsuit R_i}$, $i \in \mathbb{N}^+$. This leads to a temporal tester with $|X| = n$. The UCT is built by two states (similar to Figure 5.4) with multiple combining transitions that check whether some $x_{\diamondsuit R_i}$ does not hold and G_i holds. Let $AP' = AP \cup X$ and $\Sigma' = 2^{AP'}$. We identify the following sizes for UCT $\mathcal{U}' = \big(\Sigma', \Upsilon, Q', \{q_0'\}, \delta', \text{coBüchi}(C)\big)$:

- $|Q'| = 2$
- $|\delta'| \in O(n^2)$
- $|C'| = 1$

Thus, we observe the following sizes for the constraint systems:

- For the allrounder approach:

$$\begin{aligned} & O\big(b_\mathcal{T} \cdot (|\delta| \cdot |AP| + |I| \cdot |2^I|)\big) \\ = \; & O\Big(b_\mathcal{T} \cdot \big(n^3 \cdot 2 \cdot \log(n) + \log(n) \cdot 2^{log(n)}\big)\Big) \\ = \; & O\Big(b_\mathcal{T} \cdot \big(n^3 \cdot 2 \cdot \log(n) + \log(n) \cdot n\big)\Big) \\ = \; & O\big(b_\mathcal{T} \cdot n^3 \cdot \log(n)\big) \end{aligned}$$

- For the partial approach:

$$\begin{aligned} & O\big(b_\mathcal{T} \cdot 2^{|X|} \cdot (|\delta'| \cdot |AP'| + |I| \cdot |2^I|)\big) \\ = \; & O\big(b_\mathcal{T} \cdot 2^n \cdot (n^2 \cdot (2 \cdot \log(n) + n) + \log(n) \cdot 2^{\log(n)})\big) \\ = \; & O\big(b_\mathcal{T} \cdot n^3 \cdot 2^n\big) \end{aligned}$$

Since $O\big(b_\mathcal{T} \cdot n^3 \cdot \log(n)\big)$ is in $O\big(b_\mathcal{T} \cdot n^3 \cdot 2^n\big)$, our observation motivates the following theorem:

Theorem 10. *For the family of formulas φ as input for bounded synthesis, the size of the allrounder's constraint system is smaller than the size of the partial approach's constraint system.*

6.2 Advantage of the Partial Approach

The main advantage of our partial approach is its structured solution progress, which is desirable in theory as well as in practice. We show this statement by comparing both algorithm processes.

The allrounder uses a sequence of multiple constructions. Although Construction 2 leads to a very intuitive representation of a PLTL formula as a two-way automaton linear in the number of past-connectives in our formula, the creation of a GBW in Construction 3 already involves a lack in comprehensibility and clarity. This is caused by the idea to describe all possible proceedings in the past in a progressing way, which additionally implies an exponential blowup in the number of connectives. Finally, Construction 4 increases the missing comprehensibility and clarity of the result by multiplying another linear blowup.

However, the partial approach starts with a substitution of pure-past formulas as dedicated variables. This provides more flexibility since it allows to use any kind of construction from standard LTL to UCT. The past-subformulas itself are handled with additional constructs of just linear size in the number of past-connectives. These so-called temporal testers give an intuitive representation of the subformula in a symbolic as well as an explicit way. Thus, we are able to completely exclude the past-referring components from the automata creation. The symbolic representation is also equivalent to its encoding within the constraint system, which further reduces the workload.

The advantage can be verified exemplarily by observing the previous examples. The UCTs of the partial approach in Figure 5.4 and Figure 5.5 only consist of two states. Their idea is to stay in the rejecting state forever, whenever the underlying past requirement is violated and another variable holds true. The related temporal testers with their explicit representation in Figure 5.2 and Figure 5.3 give an intuitive description of the past-subformulas using the operator definitions. Furthermore, we presented the easy substitution of the symbolic representation in the example of Section 5.3.

6.3 Approximation Approach

The general progress of our standard approach places the temporal tester in between the implementation and the automaton. The implementation defines the input for the tester and the automaton additionally depends on the tester's outputs by taking them into its set of atomic propositions.

During the thesis, we designed an approximative approach for the future(past)-fragment. Its idea is to additionally let the implementation depend on the tester's outputs as well. To do so, we replace the inner past-

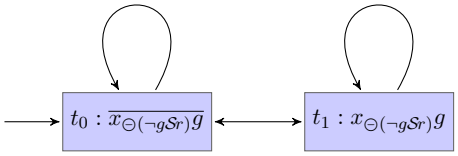


Figure 6.2: Implementation for the formula $\xi_3' = \square(g \rightarrow x_{\ominus(\neg g \mathcal{S} r)})$.

formulas for both, the implementation and the automaton, and to apply bounded synthesis on the simplified formula as usual. The truth values of the additional variables are obtained by the temporal tester in between both.

We give an example by making a substitution on formula ξ_3, i.e., $\xi_3' = \square(g \rightarrow x_{\ominus(\neg g \mathcal{S} r)})$. Rather than only building the UCT of ξ_3' in Figure 5.5 and the temporal tester in Figure 5.3, we also build an implementation that realizes ξ_3' in Figure 6.2. Thus, we have to consider $x_{\ominus(\neg g \mathcal{S} r)}$ as an additional input of our transition system, implying a cyclic dependency between tester and implementation. However, the main problem for the implementation is the time offset between the temporal tester and itself. For a past-formula that refers directly to a previous state, it has to observe a previous state of the tester, that is not visible to it. Thus, its solution requires a delayed information for the transition system referring to a single step in the past. Finding a solution to this approach would also lead to a smaller size for the transition system.

Chapter 7

Conclusion

In this thesis, we have seen three approaches for the bounded synthesis of PLTL. The first approach presents a correct solution that is not suitable in practice because of its non-elementary complexity. It describes a naive approach that operates on formula-level by applying Gabbay's Separation Theorem. Additionally, it was attached a refactored version of his theorem, whose utility is reasoned by the original sources' deviations from the standard definition and denotation of PLTL. Thus, we simplified further research on this theorem, which may also have future impact on this approach. Future work could aim on the lower bounds of the succinctness gap between LTL and PLTL, which is still left to be proven. Also, we encourage the possibility of finding alternative rules, possibly improving the non-elementary complexity.

The second presents an allrounder approach that is able to apply bounded synthesis on full PLTL. It reuses the equivalence of PLTL formulas and two-way automata, and uses a translation of these automata to non-deterministic Büchi automata with a single exponential blowup. Thus, its complexity is proven to be equivalent to the original approach of bounded synthesis. It shows that, dealing with past LTL, there is no need to only consider Gabbay's Separation Theorem. Instead of, it uses the succinctness of PLTL to deal with past-referring specifications in an efficient way. Although its complexity is exponential in the size of the input formula, there is still room for improvement. The approach totally depends on its inner construction algorithm for a UCT, which is possible to lead to a huge blowup in the number of transitions. We are open for alternative transformations from PLTL to automata.

Finally, we described a partial solution for any PLTL formula in its future(past)-fragment, which is one of the most observed fragments of PLTL. It uses a symbolic representation of its inner pure-past formulas, whose creation is not only linear in the size of the formula, but also allows a direct encoding by a quick substitution procedure. We argued that the partial

approach is preferable for reasons of a structured solution that splits its algorithm into different parts. Furthermore, it motivates the observation of other fragments in a similar way, e.g., the future$\big($past(future)$\big)$-fragment, using temporal testers. It is possible to create temporal testers for full LTL, observing so called *just-discrete systems* (JDSs), which additionally add the fairness requirement for *justice* to the symbolic representation. Finally, we also presented the idea of a fourth approach, whose further observation leads to a more efficient solution for bounded synthesis for PLTL.

Appendix A

Gabbay's Separation Theorem

Elimination 3: $\varphi = a\,\mathcal{S}\left(b \wedge \neg(\varphi_1\,\mathcal{U}\,\varphi_2)\right)$

We start by applying multiple equivalences on φ, leading again to two possibilities.

$$\begin{aligned}
\varphi \;&=\; a\,\mathcal{S}\left(b \wedge \neg(\varphi_1\,\mathcal{U}\,\varphi_2)\right)\\
&\overset{\text{Def.}}{=} a\,\mathcal{S}\left(b \wedge (\neg\varphi_1\,\mathcal{R}\,\neg\varphi_2)\right)\\
&\overset{2.2}{=} a\,\mathcal{S}\left(b \wedge \Big((\Box\neg\varphi_2) \vee \big(\neg\varphi_1\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big)\right)\\
&\overset{\text{Dist.}}{=} a\,\mathcal{S}\left(\Big(b \wedge \Box\neg\varphi_2\Big) \vee \Big(b \wedge \big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big)\right)\\
&\overset{2.3}{=} \underbrace{\Big(a\,\mathcal{S}\,(b \wedge \Box\neg\varphi_2)\Big)}_{\varphi'} \vee \underbrace{\Big(a\,\mathcal{S}\left(b \wedge \big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\right)\Big)}_{\varphi''}
\end{aligned}$$

For φ', by partitioning $\Box\neg\varphi_2$ into a past- and a future-part, one can easily obtain the following equivalent formula:

$$\varphi' = a\,\mathcal{S}\,(b \wedge \Box\neg\varphi_2) = \big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\Box(\neg\varphi_2)$$

For φ'', we can apply Elimination 1, which gives us:

$$\varphi'' = \psi_1'' \vee \psi_2'' \vee \psi_3''$$

with

- $\psi_1'' = \big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)$

- $\psi_2'' = a \,\mathcal{S}\, \Big(\ominus\big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge a \wedge \neg\varphi_1 \wedge \neg\varphi_2\Big)$
- $\psi_3'' = a \,\mathcal{S}\, (b \wedge \neg\varphi_1 \wedge \neg\varphi_2)$

Furthermore, we can combine φ' and ψ_1'', and ψ_2'' and ψ_3'', which leads to our final set of formulas:

$$\begin{aligned}\varphi \equiv_i \varphi' \vee \varphi'' &= \varphi' \vee (\psi_1'' \vee \psi_2'' \vee \psi_3'') \\ &= \underbrace{(\varphi' \vee \psi_1'')}_{\psi_1} \vee \underbrace{(\psi_2'' \vee \psi_3'')}_{\psi_2}\end{aligned}$$

with

- $\begin{aligned}\psi_1 &= \varphi' \vee \psi_1'' \\ &= \Big(\big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\Box\big(\neg\varphi_2\big)\Big) \\ &\vee \Big(\big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big) \\ &\overset{\text{Dist.}}{=} \Big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\Big) \wedge \Big(\bigcirc\Box\big(\neg\varphi_2\big) \vee \bigcirc\big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big) \\ &\overset{\text{Dist.}}{=} \Big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\Big) \wedge \bigcirc\Big(\Box\big(\neg\varphi_2\big) \vee \big(\neg\varphi_2\,\mathcal{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)\big)\Big) \\ &\overset{2.2}{=} \big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\big(\neg\varphi_1\,\mathcal{R}\,\neg\varphi_2\big) \\ &\overset{\text{Def.}}{=} \big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \bigcirc\neg\big(\varphi_1\,\mathcal{U}\,\varphi_2\big)\end{aligned}$
- $\begin{aligned}\psi_2 &= \psi_2'' \vee \psi_3'' \\ &= \Big(a \,\mathcal{S}\, \Big(\ominus\big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge a \wedge \neg\varphi_1 \wedge \neg\varphi_2\Big)\Big) \\ &\vee \Big(a \,\mathcal{S}\, (b \wedge \neg\varphi_1 \wedge \neg\varphi_2)\Big) \\ &\overset{2.3}{=} a\mathcal{S}\Big(\Big(\ominus\big((a\wedge\neg\varphi_2)\mathcal{S}(b\wedge\neg\varphi_2)\big)\wedge a\wedge\neg\varphi_1\wedge\neg\varphi_2\Big)\vee\Big(b\wedge\neg\varphi_1\wedge\neg\varphi_2\Big)\Big) \\ &\overset{\text{Dist.}}{=} a\mathcal{S}\Big(\Big(\Big(\ominus\big((a\wedge\neg\varphi_2)\mathcal{S}(b\wedge\neg\varphi_2)\big)\wedge a\wedge\neg\varphi_2\Big)\vee\Big(b\wedge\neg\varphi_2\Big)\Big)\wedge\neg\varphi_1\Big) \\ &\overset{2.1}{=} a \,\mathcal{S}\, \Big(\big((a \wedge \neg\varphi_2)\,\mathcal{S}\,(b \wedge \neg\varphi_2)\big) \wedge \neg\varphi_1\Big)\end{aligned}$

Elimination 4: $\varphi = \big(a \vee \neg(\varphi_1\,\mathcal{U}\,\varphi_2)\big)\,\mathcal{S}\,b$

As Gabbay, we will present two different approaches to handle this case. The first applies a double-negation onto the formula that includes additional elimination, whereas the second uses intuition only.

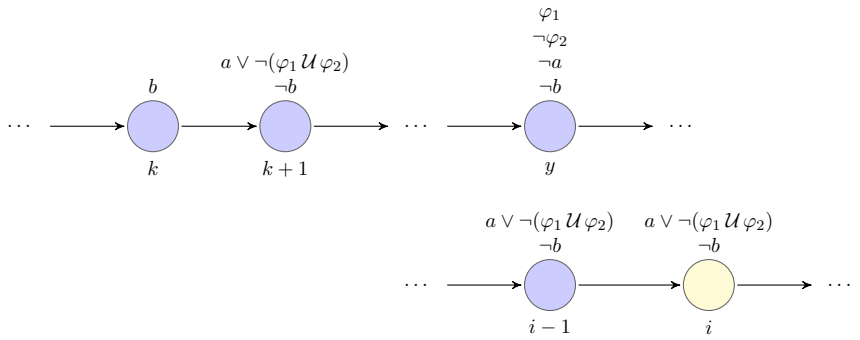
Approach 1:

We observe a double-negation on formula φ and apply several equations to observe an equivalent formula, whose rewriting asks for further eliminations.

$$
\begin{aligned}
\varphi &= \neg(\neg\varphi) \\
&= \neg\Big(\neg\Big(\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)\, \mathcal{S}\, b\Big)\Big) \\
&= \neg\Big(\neg\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)\, \mathcal{T}\, \neg b\Big) \\
&\overset{2.2}{=} \neg\Bigg(\Big(\boxminus\neg b\Big) \vee \Big(\neg b\, \mathcal{S}\, \Big(\neg\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \wedge \neg b\Big)\Big)\Bigg) \\
&\overset{\text{Morg.}}{=} \diamondminus b \wedge \neg\Big(\neg b\, \mathcal{S}\, \big(\neg a \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2) \wedge \neg b\big)\Big) \\
&\overset{\text{Morg.}}{=} \diamondminus b \wedge \underbrace{\neg\Big(\neg b\, \mathcal{S}\, \big(\neg(a \vee b) \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)\Big)}_{\text{Elimination1}}
\end{aligned}
$$

Approach 2:

For the second approach, we start with the assumptions that b was observed at some position $k < i$, and there is some "critical" position $y \in \mathbb{N}$ that does not meet a, and, thus, it has to satisfy at least $\neg\varphi_2$ by definition.



Position y triggers the following cases that has to be satisfied:

1) Whenever $\neg a$ is encountered at some position y, we have to answer it with $\neg\varphi_2$ at the same position. Furthermore, it implies that $\neg\varphi_1$ and $\neg\varphi_2$ have to be satisfied at some later position $x > y$, assuming that it is answered until i. This is ensured by the following formula:

$$\psi_1 = \Big(\neg b \wedge \Big(\neg a \rightarrow \neg\varphi_2\Big) \wedge \Big(\ominus\big((\neg b \wedge \varphi_1)\mathcal{S}(\neg a \wedge \neg b \wedge \varphi_1 \wedge \neg\varphi_2)\big) \rightarrow \neg\varphi_2\Big)\Big)\mathcal{S}b$$

2) If $\neg\varphi_2$ is not answered until position i, we have to observe $\neg(\varphi_1 \,\mathcal{U}\, \varphi_2)$ in the future:

$$\psi_2 = \big((\neg b \wedge \varphi_1) \,\mathcal{S}\, (\neg a \wedge \neg b \wedge \varphi_1 \wedge \neg\varphi_2)\big) \rightarrow \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)$$

Thus, the final rewriting is given by the conjunction of both cases:

$$\varphi \equiv_i \varphi' = \psi_1 \wedge \psi_2$$

The reader is free to choose between one of both approaches.

Elimination 5: $\varphi = \big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$

Again, we distinguish several cases:

1) Right side starts **before** i and is satisfied **at/after** i:

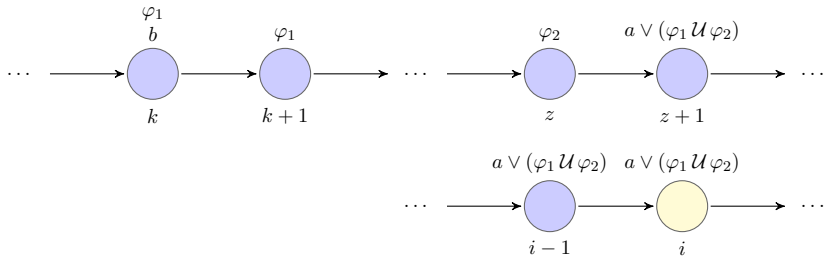
   Thus, any open request for $\varphi_1 \mathcal{U} \varphi_2$ is satisfied by a single φ_2 at position $z \in \mathbb{N}$ with $z > i$. Its formula is quite easy:

$$\psi_1 = \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big) \wedge \big(\varphi_1 \,\mathcal{U}\, \varphi_2\big)$$

2) Right side starts **at** i:

$$\psi_2 = b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

Thus, for the remaining cases, we can assume some position $z \in \mathbb{N}$ with $k \leq z < i$ such that z answers the first $\mathcal{U}$-statement of the right side.



We follow the idea of Elimination 2. In order to make a statement about the future, we distinguish whether $\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$ holds at position i, i.e., whether an $\mathcal{U}$-statement of the left side is possibly continued in position $i + 1$.

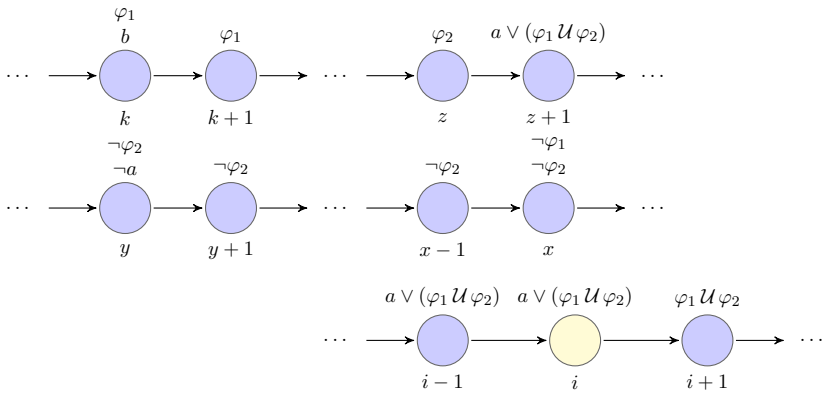
3) Right side is satisfied **before** i, $\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$ is **satisfied** at i:

   For this, we observe the negative case, i.e., there is some position $y \in \mathbb{N}$ with $z > y \geq i$ such that neither a nor $\varphi_1 \,\mathcal{U}\, \varphi_2$ are satisfied at y. We

already computed a formula for this case in Case 2 of Elimination 2, which leads us to:

$$\neg\big(a \vee (\varphi_1 \mathcal{U} \varphi_2)\big) = \underbrace{\Big((\neg a) \wedge (\Box \neg \varphi_2)\Big)}_{\psi'} \vee \underbrace{\Big((\neg a) \wedge \big(\neg \varphi_2 \,\mathcal{U}\, (\neg \varphi_1 \wedge \neg \varphi_2)\big)\Big)}_{\psi''}$$

Based on our assumption at position i, ψ' is not possible. Thus, there is some position $x \in \mathbb{N}$ with $y \leq x \leq i$, such that the right side of the $\mathcal{U}$-statement of ψ'' holds at x. Graphically:



We build a formula κ' describing the situation at x:

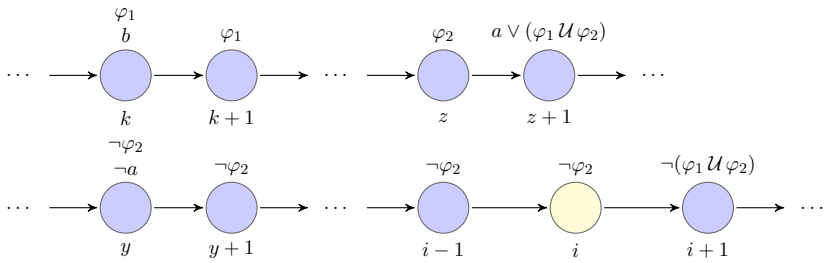
$$\alpha_x \vDash \big(\neg\varphi_2 \,\mathcal{S}\, (\neg a \wedge \neg \varphi_2)\big) \wedge \neg\varphi_1 = \kappa'$$

κ' implies a position $k < y \leq x$ such that $\sigma_y \vDash \neg a \wedge \neg(\varphi_1 \mathcal{U} \varphi_2)$. Therefore, by negating this formula, we prevent the described situation, defining our third case.

$$\psi_3 = \neg\kappa' \,\mathcal{S}\, \Big(\varphi_2 \wedge \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big)\Big) \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$$

4) Right side is satisfied **before** i, $\bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$ is **not satisfied** at i:

In here, based on the presented cases ψ' and ψ'', position $i+1$ would not continue an open answer for an $\varphi_1 \,\mathcal{U}\, \varphi_2$. Thus, we have to prevent to start a new request for φ_2 by preventing φ_1 at position i. This is done by including a slightly modified version of κ' at i. The negative situation, that we are going to prevent, is captured in the upcoming graphic, followed by the fourth formula.



$$\psi_4 = \ominus\Big(\neg\kappa' \,\mathcal{S}\, \Big(\varphi_2 \wedge \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big)\Big)\Big) \wedge \neg\Big(\neg\varphi_2 \,\mathcal{S}\, (\neg a \wedge \neg\varphi_2)\Big) \wedge \bigcirc\neg\Big(\varphi_1 \,\mathcal{U}\, \varphi_2\Big)$$

5) Unfortunately, none of the cases fully cover the formula

$$\psi_5 = \big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \varphi_2\big),$$

which needs further elimination using Elimination 2.

Thus, by building the disjunction of all cases, we observe the following elimination rule:

$$\varphi \equiv_i \varphi' = \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5$$

with

- $\psi_1 = \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big) \wedge \big(\varphi_1 \,\mathcal{U}\, \varphi_2\big)$
- $\psi_2 = b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)$
- $\psi_3 = \neg\kappa' \,\mathcal{S}\, \Big(\varphi_2 \wedge \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big)\Big) \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$
- $\psi_4 = \ominus\Big(\neg\kappa' \,\mathcal{S}\, \Big(\varphi_2 \wedge \ominus\big(\varphi_1 \,\mathcal{S}\, (b \wedge \varphi_1)\big)\Big)\Big) \wedge \neg\Big(\neg\varphi_2 \,\mathcal{S}\, (\neg a \wedge \neg\varphi_2)\Big) \wedge \bigcirc\neg\Big(\varphi_1 \,\mathcal{U}\, \varphi_2\Big)$
- $\psi_5 = \underbrace{\big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \varphi_2\big)}_{\text{Elimination 2}}$
- $\kappa' = \big(\neg\varphi_2 \,\mathcal{S}\, (\neg a \wedge \neg\varphi_2)\big) \wedge \neg\varphi_1$

Elimination 6: $\varphi = \big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$

We observe three different cases for the satisfaction of the right side. They lead to the formula

$$\varphi \equiv_i \varphi' = \psi_1 \vee \psi_2$$

with

- Right side is satisfied **at/after** i:

$$\psi_1 = \big((a \wedge \neg\varphi_2) \,\mathcal{S}\, (b \wedge \neg\varphi_2)\big) \wedge \neg\big(\varphi_1 \,\mathcal{U}\, \varphi_2\big)$$

- Right side is satisfied **before** i:

$$\psi_2 = \underbrace{\Big(a \vee (\varphi_1 \,\mathcal{U}\, \varphi_2)\Big) \,\mathcal{S}\, \Big(\big((a \wedge \neg\varphi_2) \,\mathcal{S}\, (b \wedge \neg\varphi_2)\big) \wedge \neg\varphi_1\Big)}_{\text{Elimination 2}}$$

Elimination 7: $\varphi = \big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$

Obviously, as long as $\varphi_1 \,\mathcal{U}\, \varphi_2$ of the right side is not satisfied, it is only possible to observe a of the left side. Thus, we distinguish the following cases for the satisfaction of $\varphi_1 \,\mathcal{U}\, \varphi_2$:

1) Right side satisfied **at/after** i:

$$\psi_1 = \big((a \wedge \varphi_1) \,\mathcal{S}\, (b \wedge \varphi_1)\big) \wedge \bigcirc(\varphi_1 \,\mathcal{U}\, \varphi_2)$$

2) Right side satisfied **before** i:

$$\psi_2 = \underbrace{\Big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\Big) \,\mathcal{S}\, \Big(\big(a \wedge \varphi_1\big) \,\mathcal{S}\, (b \wedge \varphi_1) \wedge a \wedge \varphi_2\Big)}_{\text{Elimination 4}}$$

3) Right side satisfied **directly before** i:

$$\psi_3 = \underbrace{\big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, (b \wedge \varphi_2)}_{\text{Elimination 4}}$$

The resulting elimination rule is given by the disjunction

$$\varphi \equiv_i \varphi' = \psi_1 \vee \psi_2 \vee \psi_3$$

Elimination 8: $\varphi = \big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big) \,\mathcal{S}\, \big(b \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$

We distinguish two cases for the satisfaction of $\neg(\varphi_1 \,\mathcal{U}\, \varphi_2)$ on the right side of φ:

1) Right side satisfied **at/after** i:

$$\begin{aligned} \psi_1 &= \Big(\big((a \wedge \neg\varphi_2) \vee \neg\varphi_2\big) \,\mathcal{S}\, \big(b \wedge \neg\varphi_2\big)\Big) \wedge \neg\Big(\varphi_1 \,\mathcal{U}\, \varphi_2\Big) \\ &= \big(\neg\varphi_2 \,\mathcal{S}\, (b \wedge \neg\varphi_2)\big) \wedge \neg\big(\varphi_1 \,\mathcal{U}\, \varphi_2\big) \end{aligned}$$

2) Right side satisfied **before** i:

$$\underbrace{\Big(a \vee \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\Big) \,\mathcal{S}\, \Big(\big(\neg\varphi_2 \,\mathcal{S}\, (b \wedge \neg\varphi_2)\big) \wedge \neg\varphi_1\Big)}_{\text{Elimination 4}}$$

The elimination rule is given by the disjunction

$$\varphi \equiv_i \varphi' = \psi_1 \vee \psi_2$$

Elimination 9: $\varphi = \ominus(\bigcirc\varphi_1)$

One may assume that this case is equivalent to φ_1. However, as φ might be observed at the initial position of a given word, we have to ensure that there "exists" a past, i.e.,

$$\varphi \equiv_i \varphi' = \ominus \mathit{true} \wedge \varphi_1.$$

Since we observe infinite words, the mirrored case does not require an additional observation, i.e.,

$$\varphi = \bigcirc\ominus\varphi_1 \equiv_i \varphi' = \varphi_1$$

Elimination 10: $\varphi = \ominus(\varphi_1 \,\mathcal{U}\, \varphi_2)$

Either φ_2 holds directly at position $i-1$, or it holds at some later position:

$$\varphi \equiv_i \varphi' = \ominus\varphi_2 \vee \big(\ominus\varphi_1 \wedge (\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$$

Elimination 11: $\varphi = \ominus\neg(\varphi_1 \,\mathcal{U}\, \varphi_2)$

Either $\neg\varphi_1$ and $\neg\varphi_2$ hold at position $i-1$, or both are satisfied at same later position:

$$\varphi \equiv_i \varphi' = \ominus\big(\neg\varphi_1 \wedge \neg\varphi_2\big) \vee \big(\ominus\neg\varphi_2 \wedge \neg(\varphi_1 \,\mathcal{U}\, \varphi_2)\big)$$

Elimination 12: $\varphi = a \,\mathcal{S}\, (b \wedge \bigcirc \varphi_1)$

Either the right side holds directly, or we have to observe a "shifted" case:

$$\varphi \equiv_i \varphi' = \big(a \,\mathcal{S}\, (\ominus b \wedge a \wedge \varphi_1)\big) \vee \big(b \wedge \bigcirc \varphi_1\big)$$

Elimination 13: $\varphi = (a \vee \bigcirc \varphi_1) \,\mathcal{S}\, b$

Again, we observe a "shifted" case, if b does not hold at the current position:

$$\varphi \equiv_i \varphi' = \Big(\big((\ominus a \vee \varphi_1) \,\mathcal{S}\, \ominus b\big) \wedge \big(a \vee \bigcirc \varphi_1\big)\Big) \vee b$$

Elimination 14: $\varphi = (a \vee \bigcirc \varphi_1) \,\mathcal{S}\, (b \wedge \bigcirc \varphi_1)$

Finally, in this case, we have to observe the "shifting" twice:

$$\varphi \equiv_i \varphi' = \Big(\big((\ominus a \vee \varphi_1) \,\mathcal{S}\, (\ominus b \wedge \varphi_1)\big) \wedge \big(a \vee \bigcirc \varphi_1\big)\Big) \vee \Big(b \wedge \bigcirc \varphi_1\Big)$$

Proof Idea of Gabbay's Separation Theorem

In order to prove that an elimination terminates, Gabbay gives multiple definitions and lemmas. Rather than giving the full proof, we present its idea in partial detail.

Gabbay starts by observing a given PLTL formula as a binary tree, where its *degree* is defined as the number of *nestings* between past-operators and future-operators. Thus, given such a nesting with operator o_2 in the scope of operator o_1, we can observe the number of operators **above** o_2, up to o_1, and the number of operators **below** o_2. The proof is done by induction, observing three values:

- The induction is done on the number of nestings n.
- The number of operators above o_2, up to o_1, decreases with each applied elimination and / or equation.
- The number of operators below o_2 increases with each applied elimination and / or equation.

The elimination terminates if the number of operators between o_1 and o_2 is equal to 0, i.e., o_2 is out of the scope of o_1, and the number of operators below o_2 increases, i.e., the elimination causes additional operators as cost for the elimination of nestings, without increasing the number of nestings itself. If the number of nestings is equal to 0, the algorithm terminates.

Bibliography

[1] Rajeev Alur and Salvatore La Torre. Deterministic Generators and Games for LTL Fragments. *ACM Trans. Comput. Logic*, 5(1):1–25, January 2004.

[2] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller Synthesis for Timed Automata1. *IFAC Proceedings Volumes*, 31(18):447–452, 1998.

[3] Marco Benedetti and Alessandro Cimatti. Bounded Model Checking for Past LTL. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'03, pages 18–33, Berlin, Heidelberg, 2003. Springer-Verlag.

[4] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of Reactive(1) Designs. *J. Comput. Syst. Sci.*, 78(3):911–938, May 2012.

[5] Paola Bonizzoni and Giancarlo Mauri. On Automata on Infinite Trees. *Theoretical Computer Science*, 93(2):227–244, 1992.

[6] Alonzo Church. Logic, Arithmetic, and Automata. In *Proc. 1962 Int. Congr. Math. Upsala*, pages 23–25, 1963.

[7] Alessandro Cimatti, Marco Roveri, and Daniel Sheridan. Bounded Verification of Past LTL. In *Formal Methods in Computer-Aided Design*, pages 245–259, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] Edmung M. Clare, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer International Publishing AG, Cham, Switzerland, June 2018.

[9] Bernd Finkbeiner and Sven Schewe. Bounded Synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):519–539, 2013.

[10] Dov M. Gabbay. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In *Temporal Logic in Specification*, pages 409–448, London, UK, UK, 1987. Springer-Verlag.

[11] Paul Gastin and Denis Oddoux. Fast LTL to Büchi Automata Translation. In *Computer Aided Verification*, pages 53–65, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[12] Paul Gastin and Denis Oddoux. LTL with Past and Two-Way Very-Weak Alternating Automata. In *Mathematical Foundations of Computer Science 2003*, pages 439–448, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[13] Erich Grädel, Wolfgang Thomas, and Wilke Thomas. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[14] Johan Anthony Wilem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[15] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic Verification of Linear Temporal Logic Specifications. In *Automata, Languages and Programming*, pages 1–16, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[16] Orna Kupferman and Moshe. Y. Vardi. Safraless Decision Procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 531–540, October 2005.

[17] 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, Washington, DC, USA, 2002. IEEE Computer Society.

[18] François Laroussinie and Philippe Schnoebelen. A Hierarchy of Temporal Logics with Past. *Theor. Comput. Sci.*, 148(2):303–324, September 1995.

[19] Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi Junttila. Simple is Better: Efficient Bounded Model Checking for Past LTL. In *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'05, pages 380–395, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The Glory of the Past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, UK, 1985. Springer-Verlag.

[21] Nicolas Markey. Temporal Logic with Past is Exponentially More Succinct. *EATCS Bulletin*, 79:122–128, 2003.

[22] Nir Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 255–264, August 2006.

[23] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, October 1977.

[24] Matteo Pradella, Pierluigi San Pietro, Paola Spoletini, and Angelo Morzenti. Practical Model Checking of LTL with Past. In *Proceedings of the 1st International Workshop on Automated Technology for Verification and Analysis*, ATVA03, January 2003.

[25] Shmuel Safra. On the complexity of ω-automata. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, October 1988.

[26] Sven Schewe. Tighter Bounds for the Determinisation of Büchi Automata. In *Foundations of Software Science and Computational Structures (FoSSaCS 2009)*, volume 5504, pages 167–181, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[27] Aravinda P. Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, July 1985.