Contribution to Efficient Büchi Game-Solving for SPOT

Rataud Quentin
(supervisor: Philipp Schlehuber-Caissier)

Technical Report n^o202407-techrep-rataud, July 2024
revision b77f4b3

In the domain of reactive system verification, efficient solving of Büchi games is critical for ensuring correctness against temporal logic specifications. This paper focuses on enhancing SPOT's solver by specializing in Büchi games, specifically transitional variants, to address performance bottlenecks associated with exponential complexity. By introducing improvements to existing algorithms and optimizations tailored to Büchi games, including strategies for SCC decomposition and specialized data structures, significant improvements in solving times and scalability are achieved.

Dans le domaine de la vérification des systèmes réactifs, la résolution efficace des jeux de Büchi est cruciale pour garantir la correction par rapport aux spécifications de logique temporelle. Ce document se concentre sur l'amélioration du solveur de SPOT en se spécialisant dans les jeux de Büchi, notamment les variantes transitionnelles, afin de résoudre les goulots d'étranglement de performance associés à une complexité exponentielle. En introduisant des améliorations aux algorithmes existants et des optimisations adaptées aux jeux de Büchi, incluant des stratégies de décomposition des SCC (*Strongly Connected Components*) et des structures de données spécialisées, des améliorations significatives des temps de résolution et de la scalabilité sont obtenues.

Keywords
Büchi games, reactive systems, model checking, game theory, automata theory

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
qrataud@lre.epita.fr – http://www.lre.epita.fr/

Copying this document

Copyright © 2023 LRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being just “Copying this document”, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is provided in the file COPYING.DOC.

Contents

Chapter 1

Introduction

This paper addresses the challenge of optimizing SPOT's solver for Büchi games, crucial in verifying temporal logic properties. Büchi games are fundamental in modeling systems that operate indefinitely, requiring precise analysis of infinite sequences of system behaviors against specified requirements.

Problem Statement The existing approach in SPOT utilizes a generic solver that converts LTL formulas into parity games, employing algorithm from Zielonka (1998). While effective in many cases, this method encounters exponential complexity under certain conditions, leading to performance bottlenecks. Specifically, Büchi games, a subset of parity games, demand tailored strategies for efficient resolution due to the simplicity of their acceptance condition.

This research aims to enhance SPOT's solver by specializing the solver for Büchi games.

Contributions In this paper, we present several key contributions:

- **Specialized Büchi Game Solvers**: Development and implementation of specialized solvers tailored for Büchi games, focusing on transitional Büchi game variants.
- **Algorithmic Enhancements**: Introduction of novel algorithms that optimize the solving process by leveraging structural insights into Büchi games. This includes efficient decomposition strategies for strongly connected components (SCCs) and novel data structures to expedite computation.

Guide for Future Students Additionally, this paper serves as a guide for future students joining the LRE The concepts and methodologies described herein are aligned closely with the conventions used by SPOT, the solver developed by the LRE. By providing comprehensive coverage of necessary theories and techniques, this paper equips students with the foundational knowledge needed to contribute effectively to `ltlsynt`, the synthesis solver of SPOT. It aims to ensure continuity and ease of transition for new researchers aiming to build upon or extend the capabilities of SPOT in the domain of reactive system verification.

Overview of the Paper The remainder of this paper is structured as follows:

- Chapter 2 introduces foundational concepts in automata theory and game theory essential for understanding Büchi games and their applications in reactive system verification.
- Chapter 3 delves into the theory and strategic aspects of Büchi games, highlighting their relevance in SPOT and outlining traditional solving algorithms.
- Chapter 4 presents our novel approaches to enhancing Büchi game solvers, focusing on SCC decomposition and advanced data structures for improved efficiency.

Acknowledgments Before proceeding, we acknowledge the invaluable feedback and contributions from our reviewers, whose insights have significantly enriched this work.

Chapter 2

Preliminaries

This chapter introduces foundational concepts in automata theory and game theory that form the basis for subsequent discussions in this paper.

Section 2.1.1 introduces deterministic finite automata (DFA), which are fundamental mathematical models used to represent systems processing sequences of inputs. DFA are characterized by their ability to transition between states based on input symbols, leading to specific acceptance criteria for input sequences.

Section 2.1.2 introduces ω-automata, which handle infinite inputs and are crucial in modeling reactive systems that continuously interact with their environments. ω-automata extend the capabilities of DFA by introducing acceptance conditions adequate for infinite runs of input sequences.

Section 2.1.3 then introduces Transitional Emerson-Lei Automata (TELA), which refine ω-automata to specify acceptance conditions using Boolean formulas over sets of states visited finitely or infinitely often. TELA are particularly relevant in contexts such as reactive synthesis, where verification and synthesis tasks demand precise handling of temporal logic constraints.

Then, Section 2.1.4 discusses Büchi automata, a special case of TELA with acceptance conditions based on infinite occurrences of specific patterns within input sequences.

Finally, Section 2.2 defines key concepts in two-player game theory, introducing games played on structures akin to complete ω-automata, where states are partitioned between two players: the environment and the controller.

2.1 Automata

2.1.1 Deterministic Finite Automata

An automaton is a mathematical model used to represent systems that process sequences of inputs over discrete time steps. When given a sequence of inputs, called a word, an automaton processes these inputs one symbol at a time. Each symbol is chosen from a set called the *input alphabet*. At any moment during its operation, the automaton is in one of its states. As it receives each input symbol, it transitions to a new state according to a transition function, which determines the next state based on the current state and input symbol. The sequence of states through which the automaton passes as it processes the input word is called a *run*.

In the context of Spot, the behavior of a controller can be represented as transitions within an automaton, where each input corresponds to an action or signal received by the controller, and each state represents a possible configuration of the system.

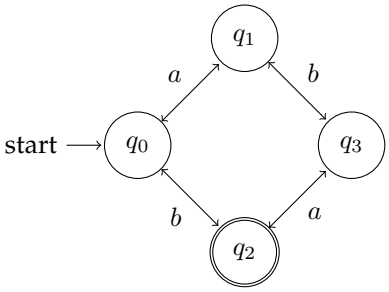

Figure 2.1: Example of Deterministic Finite Automaton

Definition 2.1 (Deterministic Finite Automaton (DFA))**.** *A* Deterministic Finite Automaton *is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where:*

- *Q is a finite set of states.*
- *Σ is a finite set of symbols, called the* alphabet*.*
- *$\delta : Q \times \Sigma \to Q$ is a transition function.*
- *$q_0 \in Q$ is the initial state.*
- *$F \subseteq Q$ is the set of accepting states.*

A transition $\delta(q, a) = q'$ indicates that the automaton moves from state q to state q' upon reading the input symbol a. Figure 2.1 shows an example of finite automaton, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\delta(q_0, a) = q_1$, and $F = \{q_2\}$.

Definition 2.2 (Run on Finite Automaton)**.** *A* run *of an automaton on a finite* word $w = a_0a_1a_2 \dots a_{n-1} \in \Sigma^*$ *is a sequence of states $r = q_0, q_1, q_2, \dots, q_n$ such that $\delta(q_i, a_i) = q_{i+1}$ for all $i \geq 0$. The word w is* accepted *by the automaton if the run ends in an accepting state, i.e., $q_n \in F$ for the last state q_n of the run.*

The automaton in Fig. 2.1 shows an example of finite automaton accepting any word containing an even number of a and an odd number of b. For instance, on this automaton, the run on the word $abbab$ is the sequence $q_0, q_1, q_3, q_1, q_0, q_2$. Since q_2 is an accepting state, the word $abbab$ is *accepted* by the automaton.

Definition 2.3 (Complete Automaton)**.** *An automaton is* complete *if all states always have at least one outgoing transition for any given letter. More formally, an automaton $A = (Q, \Sigma, \delta, q_0, F)$ is said to be complete if for any state $s_1 \in Q$ and any letter $a \in \Sigma$, there exists a state $s_2 \in Q$ such that $(s_1, a, s_2) \in \delta$.*

2.1.2 ω-Automata

Many of today's problems in computer science involve systems that do not simply terminate after processing a finite amount of data, but rather must continuously interact with their environment. Consider a smart traffic light system that continually receives sensor data about waiting cars and adjusts its signals accordingly. Such systems are reactive, since they must respond continuously and adaptively to changing inputs without ever terminating.

Unlike finite automata, ω-automata handle infinite inputs. They process infinite words, $w \in \Sigma^\omega$, using specific acceptance conditions to determine whether an infinite run is accepted.

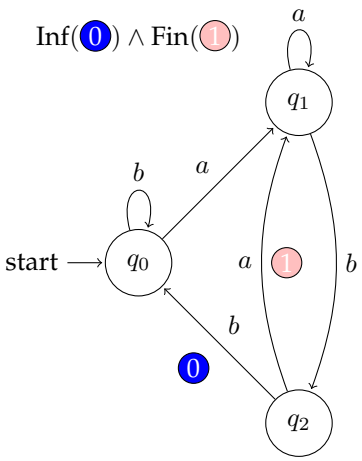

Figure 2.2: Example of Transitional Emerson-Lei Automaton

Definition 2.4 (ω-Automaton)**.** *An* ω-automaton *is defined as a quintuple* $A = (Q, \Sigma, \delta, q_0, \alpha)$*, where* Q*,* Σ*,* δ*, and* q_0 *are defined as in the Deterministic Finite Automaton, and* $\alpha \subseteq \Sigma^\omega$ *is the acceptance set.*

Definition 2.5 (Run on ω-automaton)**.** *A* run *of an* ω*-automaton on an infinite word* $w = a_0a_1a_2 \ldots \in \Sigma^\omega$ *is an infinite sequence of states* $r = q_0, q_1, q_2, \ldots$ *such that* $\delta(q_i, a_i) = q_{i+1}$ *for all* $i \geq 0$*. The run is* accepted *if it belongs to the acceptance set* α*.*

2.1.3 Transitional Emerson-Lei Automata

Transitional Emerson-Lei automata (TELA) extend ω-automata by specifying acceptance conditions using positive Boolean formulas over sets of states visited finitely or infinitely often.

These automata are particularly relevant in the context of reactive synthesis tools like Spot, which convert sets of Linear Temporal Logic (LTL) constraints into automata for verification and synthesis tasks. The constraints typically yield specific acceptance conditions, such as combinations of "infinitely often" and "finitely often" behaviors, which are precisely captured by TELA.

A *Transitional Emerson-Lei Automaton* is an ω-automaton with an acceptance condition defined as the conjunction of Inf(m) (the set of all runs where the mark m appears infinitely often) and Fin(m) (the set of all runs where the mark m appears finitely often) conditions for various marks m.

Definition 2.6 (Transitional Emerson-Lei Automaton (TELA))**.** *Formally, a* Transitional Emerson-Lei Automaton *is a tuple* $A = (Q, M, \Sigma, \delta, q_0, \alpha)$*, where:*

- Q *is a finite set of states.*
- M *is a finite set of marks (or colors).*
- Σ *is a finite input alphabet.*
- $\delta \subseteq Q \times \Sigma \times 2^M \times Q$ *is a finite set of transitions.*
- $q_0 \in Q$ *is the initial state.*
- $\alpha \subseteq \delta^\omega$ *is the acceptance condition, defined as a Boolean combination of conditions Inf*(m) *(mark* m *appears infinitely often) and Fin*(m) *(mark* m *appears finitely often), where* $m \in M$*.*

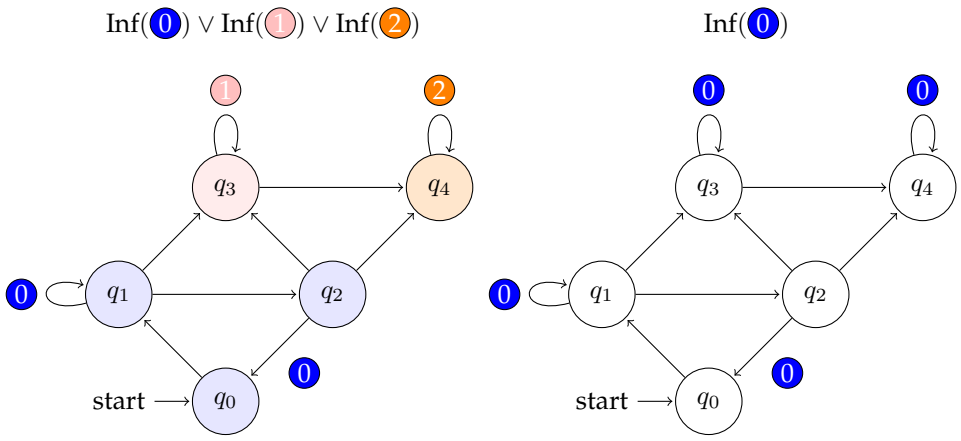

Figure 2.3: Example of weak Transitional Emerson-Lei Automaton and its corresponding Büchi Automaton

Figure 2.2 shows an example of Transitional Emerson-Lei Automaton, where $Q = \{q_0, q_1, q_2\}$, $M = \{0, 1\}$, $\Sigma = \{a, b\}$, $(q_2, b, \{0\}, q_0) \in \delta$, and $\alpha = \text{Inf}(0) \wedge \text{Fin}(1)$.

A run of a TELA is an infinite sequence of transitions, and the acceptance condition α is interpreted over the set of marks seen infinitely often during the run.

The Transitional Emerson-Lei Automaton in Fig. 2.2 accepts all words that contains infinitely many times the subsequence abb, but finitely many times the subsequence aba.

2.1.4 Büchi Automata

A *weak* TELA is one in which all transitions within a strongly connected component share the same color. These automata can always be transformed into an automaton with the acceptance condition Inf(0). Figure 2.3 shows an example of a weak TELA. The states can be partitioned into three distinct strongly connected components:

- $\{q_0, q_1, q_2\}$, containing only mark 0;
- $\{q_3\}$, containing only mark 1;
- $\{q_4\}$, containing only mark 2.

In the context of Spot, many simple formulas yield weak Transitional Emerson-Lei Automata.

Definition 2.7 (Transitional Büchi Automaton)**.** *A Transitional Büchi Automaton is a special case of a TELA where the acceptance condition is Inf(m) for a particular mark m.*

Formally, it is defined as $A = (Q, \Sigma, \delta, q_0, B)$, where:

- *Q is a finite set of states.*
- *Σ is a finite input alphabet.*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a finite set of transitions.*
- *$q_0 \in Q$ is the initial state.*
- *$B \subseteq \delta$ is the set of Büchi transitions, that is, the set of all transitions marked with m.*

A run of a Transitional Büchi Automaton is accepting if it traverses some Büchi transitions infinitely often.

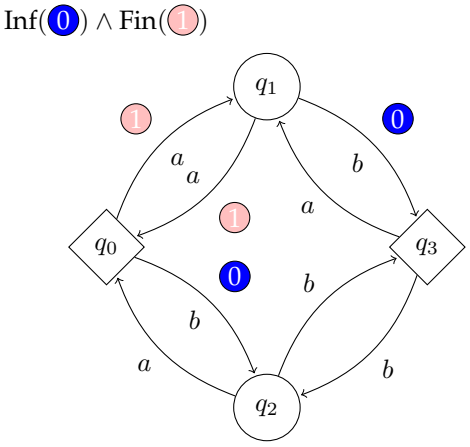

Figure 2.4: Example of a two-player game

2.2 Two-Player Games

2.2.1 Games

In the context of automata theory, a game is played on a structure that can be viewed as a special kind of complete ω-automaton, where states are partitioned between two players: the environment and the controller. Formally, a game is defined as follows:

Definition 2.8 (Game)**.** *A* game *is a tuple* $G = (Q_0, Q_1, \Sigma, \delta, \alpha)$*, where:*

- Q_0 *is a finite set of states belonging to the environment.*
- Q_1 *is a finite set of states belonging to the controller.*
- $Q = Q_0 \cup Q_1$ *usually denotes the set of all states.*
- Σ *is a finite set of input symbols (the alphabet).*
- $\delta : Q \times \Sigma \to Q$ *is the transition function.*
- $\alpha \subseteq Q \times \Sigma^\omega$ *is the acceptance condition.*

Note that in this definition, a game does not necessarily include an initial state. This is because in this paper, we are mainly interested uniform games, where we want to solve games for all possible initial states. Following the conventions of Spot in this paper, the states in Q_0 are usually represented by circles, and the state in Q_1 are represented by diamonds.

Figure 2.4 shows an example of a game, where $Q_0 = \{q_1, q_2\}$, $Q_1 = \{q_0, q_3\}$, $\Sigma = \{a, b\}$, $\delta(q_0, a) = q_1$, and α is the set of all infinite words containing the subsequence ab infinitely many times and containing the sequence aa finitely many times.

Definition 2.9 (Play)**.** *A* play $\rho = (q_0, w) \in Q \times \Sigma^\omega$ *in the game is akin to a run in an ω-automaton from initial state q_0, where each transition of the infinite word is chosen either by the environment or the controller, depending on the current state. Specifically, if the current state is in Q_0, the environment decides the next transition, otherwise, the controller makes the decision.*

The acceptance condition α determines if a play is accepted or not by the game. This condition specifies the set of plays that are accepted. Usually, the controller (Player 1) works towards having the play accepted, whereas the environment (Player 0) works towards having the play

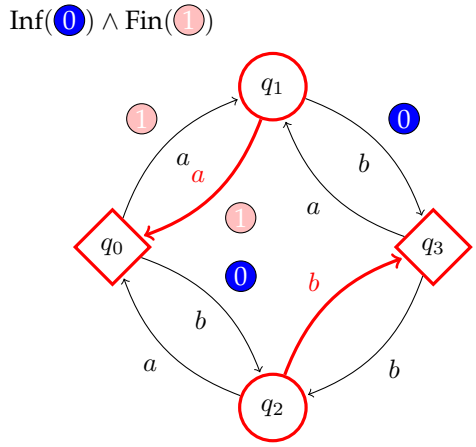

Figure 2.5: Example of a winning positional strategy for the environment.

not accepted. If a play ρ satisfies the acceptance condition α ($\rho \in \alpha$), then we can say that the controller is winning, otherwise, the environment is winning.

In the game depicted in Fig. 2.4, the two players will take turn outputting a letter. The controller succeeds if the resulting play contains the subsequence ab infinitely many times and the sequence aa finitely many times.

2.2.2 Strategies

The concept of *strategies* determines the behavior of each player during the game. A strategy prescribes how a player should move through the game, potentially based on the history of the play so far.

Definition 2.10 (Strategy)**.** *Formally, a* strategy $\sigma : Q \times \Sigma^* \to \Sigma$ *for Player* $i \in \{0, 1\}$ *(where* i *is either* 0 *for the environment or* 1 *for the controller) is a function defined for any initial state* $q_0 \in Q$ *and any ongoing play* $w \in \Sigma^*$ *that started from* q_0 *and is currently in a state* $q \in Q_i$*. The function then returns the symbol* a *from the transition* $\delta(q, a) = q'$ *the player* i *will select.*

By convention, we denote strategies for the controller by σ and counter-strategies for the environment by τ. The play induced by two strategies σ and τ starting from an initial state $q_0 \in Q$ is the sequence where

$$\rho_{n+1} = \begin{cases} \tau(q_0, \rho_0\rho_1 \dots \rho_n) & \text{if } \rho \text{ ends in } Q_0 \\ \sigma(q_0, \rho_0\rho_1 \dots \rho_n) & \text{if } \rho \text{ ends in } Q_1 \end{cases}$$

While the general definition of a strategy allows it to depend on the entire history of the play, in practice, we often consider simpler forms of strategies known as *positional strategies*. A positional strategy makes its decisions based only on the current state, ignoring the history of the play.

Definition 2.11 (Positional Strategy)**.** *Formally, a strategy σ is* positional *if for any ongoing play $w \in \Sigma^*$ starting from the initial state $q_0 \in Q$ and currently ending on state q_n, $\sigma(q_0, w) = \sigma(q_n)$.*

Thus, a positional strategy for Player i can be represented as a function $\sigma : Q_i \to \Sigma$. We typically denote such positional strategies in this manner. Following Spot conventions, positional strategies are represented by coloring the chosen transition $\tau(q_0)$ in red for all states $q_0 \in Q_0$, and coloring the chosen transition $\sigma(q_1)$ in green for all states $q_1 \in Q_1$. Figure 2.5 shows the positional strategy chosen by the environment. Whenever the active state is q_1, the environment will produce the letter a, and whenever the active state is q_2, the environment will produce the letter b.

Definition 2.12 (Winning Strategy)**.** *We say that a strategy σ is winning from a state $q_0 \in Q$ for Player $i \in \{0, 1\}$ if, for all counter-strategy τ for player $1 - i$, the play induced by σ and τ is still winning for Player i.*

The set of all states where Player i has a winning strategy is denoted as his *winning region* W_i.

Theorem 2.13. *For any game $G = (Q_0, Q_1, \Sigma, \delta, \alpha)$, $W_0 \cap W_1 = \emptyset$.*

Proof. Suppose there exists a state $q \in W_0 \cap W_1$. Then there exist a winnning strategy τ from state q for the environment, and a winning strategy σ from state q for the controller. Let ρ be the play induced by τ and σ from state q. Since τ is winning from state q for the environment, we must have $\rho \notin \alpha$, and since σ is winning from state q for the controller, we must have $\rho \in \alpha$, thus leading to a contradiction. □

Interstingly, we do not necessarily have $W_0 \cup W_1 = Q$. We denote a game where $W_0 \cup W_1 \neq Q$ as *undetermined*. Following Spot conventions, the winning regions are represented by coloring the states in red for all states $q_0 \in W_0$, and in green for all states $q_1 \in W_1$. Since the strategy depicted in Fig. 2.5 is always winning for the environment no matter the initial state, all the states belongs to the winning region of the environment.

Chapter 3

Büchi Games

In this chapter, we will focus into the theory and applications of Büchi games, with a specific focus on the transitional Büchi game variant. Transitional Büchi games are particularly relevant in the context of Spot, where most of the computation is done on transition-based structures. This focus allows us to explore the strategic aspects of gameplay and the computational methods employed to analyze and solve Büchi games efficiently.

Section 3.1 defines the foundational concepts of Büchi games, including the transition system, Büchi conditions, and the game structure itself. We introduce strategies employed by both players — the controller and the environment — to achieve their respective objectives within the game. Key theoretical constructs such as the controlled predecessor and attractor sets are then explored in detail, providing tools for determining optimal strategies and analyzing game outcomes.

Then, Section 3.2 will introduce the traditional algorithm to solve any Büchi game in $O(nm)$ time, determining optimal positional strategies for both players, for all states of the game.

3.1 Concepts

A Büchi game is a type of two-player game where the objective is to satisfy a Büchi condition. In a standard Büchi Game, the controller aims to ensure that a specific subset of states is visited infinitely often. However, as most of the computations done in Spot are transition-based, we will focus on a variant called *Transitional Büchi Game*, [1] where the controller aims to ensure that a set of *transitions* is visited infinitely often, while the environment tries to prevent this.

Formally, a transitional Büchi game is defined as follows:

Definition 3.1 (Transitional Büchi Game)**.** *A* transitional Büchi game *is a tuple* $G = (Q_0, Q_1, \delta, B)$*, where:*

- Q_0 *and* Q_1 *are the sets of states controlled by the controller and the environment, respectively,*
- $\delta \subseteq Q \times Q$ *is the transition relation,*
- $B \subseteq \delta$ *is the set of Büchi transitions.*

Figure 3.1 shows an example of a transitional Büchi game, where $Q_0 = \{q_0, q_3, q_4\}$, $Q_1 = \{q_1, q_2, q_5\}$, $(q_1, q_3) \in \delta$ and $(q_1, q_2) \in B$.

[1] Since this paper will only focus on transitional Büchi games rather than standard ones, all future mentions of a *Büchi game* in this paper will refer to the transitional variant.

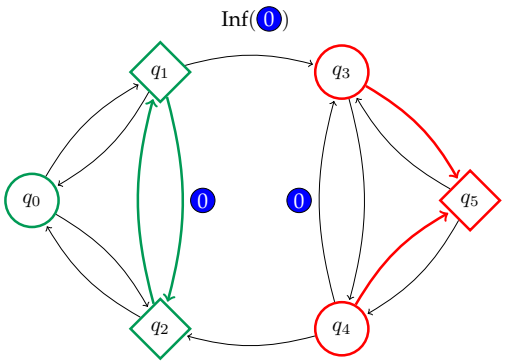

Figure 3.1: Example of a Büchi game

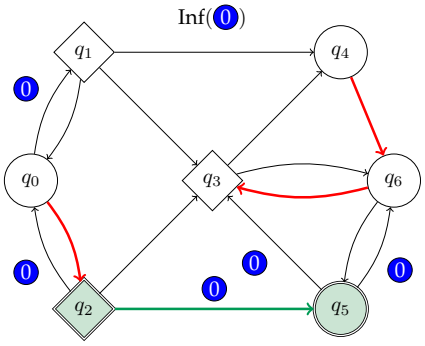

Figure 3.2: Controlled Predecessor of the Büchi transitions for the controller

A *run* in a Büchi game is an infinite sequence of states $q_0, q_1, q_2, \ldots$ such that $(q_i, q_{i+1}) \in \delta$ for all $i \geq 0$. The controller wins the game if the set of Büchi transitions B appears infinitely many times in the run. Otherwise, the environment wins by preventing this condition from being satisfied. Following the strategy σ highlighted in green on Fig. 3.1 and starting from the state q_1 will produce the run $q_1, q_2, q_1, q_2, \ldots$. Since the transition $(q_1, q_2) \in B$ is a Büchi transition, this run will be winning for the controller.

In the definition, δ is a transition relation, meaning it is a set of ordered pairs of states. This abstraction removes the need to consider an alphabet or symbols associated with the transitions, which simplifies the structure. Considering multiple transitions between the same pair of states is unnecessary because the controller always prefers marked transitions, while the environment can always choose unmarked ones. Therefore, reducing δ to a simple set of transitions without additional symbols makes the model less heavy and focuses on the strategic aspect of the game.

3.1.1 Controlled Predecessor

The following definition of controlled predecessor is an extension of the one that can be found in Zimmermann et al. (2016) for transitional games.

Definition 3.2 (Controlled Predecessor (CPre)). *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider a set $S \subseteq Q \cup \delta$ of states and transitions. A state q belongs to the* Controlled Predecessor *of S for Player i, denoted as $\mathrm{CPre}(S)$, if:*

- *$q \in Q_i$ and q has at least one outgoing transition (q, q') such that $q' \in S$ or $(q, q') \in S$;*
- *$q \in Q_{1-i}$ and q only has outgoing transitions (q, q') such that $q' \in S$ or $(q, q') \in S$.*

Figure 3.2 shows a Büchi game were $\mathrm{CPre}_1(B) = \{q_2, q_5\}$. q_5 belongs to the controlled predecessor since all of the outgoing transitions from q_5 are Büchi transitions. q_2 belongs to the controlled predecessor since there is at least one outgoing transition that is a Büchi transition. For all states in the controlled predecessor belonging to the controller, an example of strategy σ guaranteeing that a Büchi transition will be taken next turn is highlighted in green. Similarly, for each state that is not in the controlled predecessor belonging to the environment, an example of counter-strategy τ guaranteeing that no Büchi transition will be taken next turn is highlighted in red.

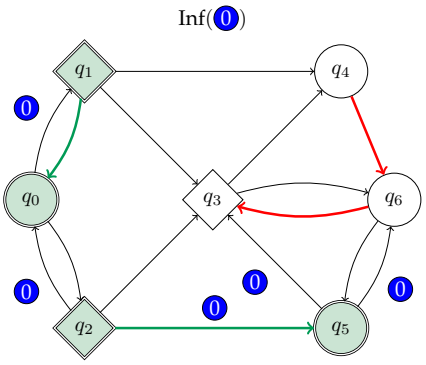

Figure 3.3: Attractor of the Büchi transitions for the controller

Theorem 3.3. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider any set $S \subseteq Q \cup \delta$ of states and transitions. Then, there exist a positional strategy for Player i from any state q forcing the next move to either take a transition in S or set the next active state in S if and only if $q \in \mathrm{CPre}_i(S)$.*

Proof. Consider any set S of states and transitions in G, and consider any state q in Q. Then,

- If $q \in \mathrm{CPre}_i(S) \cap Q_i$, then q has a successor q' such that $(q, q') \in S$ or $q' \in S$. Setting $\sigma(q) = q'$ will either directly take a transition in S or set the next active state in S.
- If $q \in \mathrm{CPre}_i(S) \cap Q_{1-i}$, then q only has successors q' such that $(q, q') \in S$ or $q' \in S$. No matter the counter-strategy, the active state will either directly take a transition in S or set the next active state in S.
- If $q \in \overline{\mathrm{CPre}_i(S)} \cap Q_i$, then q must have no successor nor outgoing transitions in S. No matter our strategy, we cannot directly take a transition in S nor set the next active state in S.
- Finally, if $q \in \overline{\mathrm{CPre}_i(S)} \cap Q_{1-i}$, then q must have a successor q' such that $(q, q') \notin S$ and $q' \notin S$. Setting $\tau(q) = q'$ as a counter-strategy proves that no strategy will allow Player i to force the next active state in S or force to directly take a transition in S.

□

3.1.2 Attractor

More generally, the attractor of a set of states or transitions S for a player i describes the set of states from which Player i has a strategy to either set the active state in S or eventually take a transition in S in a finite amount of moves. The following definition is also an extension of the one that can be found in Zimmermann et al. (2016) for transitional games.

Definition 3.4 (Attractor (Attr))**.** *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider any set $S \subseteq Q \cup \delta$ of states and transitions.*

The attractor $\mathrm{Attr}_i(R)$ *of the set S is defined inductively as follows:*

$$\mathrm{Attr}_i^0(S) = S \cap Q,$$
$$\mathrm{Attr}_i^{n+1}(S) = \mathrm{Attr}_i^n(S) \cup \mathrm{CPre}_i(\mathrm{Attr}_i^n(S) \cup S),$$
$$\mathrm{Attr}_i(S) = \bigcup_{n \in \mathbb{N}} \mathrm{Attr}_i^n(S).$$

Intuitively, $\mathrm{Attr}_i^n(S)$ describes the set of all states from which Player i can force to eventually take a transition in S or reach a state in S in no more than n steps.

Figure 3.3 shows the attractor $\mathrm{Attr}_1(B)$ of the Büchi transitions for the controller. The step by step computation of $\mathrm{Attr}_1(B)$ would be as follows:

- $\mathrm{Attr}_1^0(B) = B \cap Q = \emptyset$, meaning the controller cannot start on a state that is already marked, since there is none;
- $\mathrm{Attr}_1^1(B) = \mathrm{Attr}_1^0(B) \cup \mathrm{CPre}_1(\mathrm{Attr}_1^0(B) \cup B) = \mathrm{CPre}_1(B) = \{q_2, q_5\}$, meaning the controller can force to immediately take a transition in B from state q_2 or q_5;
- $\mathrm{Attr}_1^2(B) = \mathrm{Attr}_1^1(B) \cup \mathrm{CPre}_1(\mathrm{Attr}_1^1(B) \cup B) = \{q_2, q_5\} \cup \mathrm{CPre}_1(\{q_2, q_5\} \cup B) = \{q_0, q_2, q_5\}$, meaning the controller can force to take a transition in B in at most two steps from state q_0, q_2 or q_5. Notice the importance of the union of states and transitions as parameter to the CPre function, as q_0 only belongs to the controlled predecessor since it has the choice between taking a transition in B to q_1, or taking a transition not in B to q_2;
- $\mathrm{Attr}_1^3(B) = \mathrm{Attr}_1^2(B) \cup \mathrm{CPre}_1(\mathrm{Attr}_1^2(B) \cup B) = \{q_0, q_2, q_5\} \cup \mathrm{CPre}_1(\{q_0, q_2, q_5\} \cup B) = \{q_0, q_1, q_2, q_5\}$, meaning the controller can force to take a transition in B in at most three steps from state q_0, q_1, q_2, or q_5;
- Finally, $\mathrm{Attr}_1^4(B) = \mathrm{Attr}_1^3(B) \cup \mathrm{CPre}_1(\mathrm{Attr}_1^3(B) \cup B) = \mathrm{Attr}_1^3(B)$. It can be shown that no additional states will ever enter the attractor.

Theorem 3.5. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider any set $S \subseteq Q \cup \delta$ of states and transitions. Then, there exist a positional strategy for player i that will guarantee to visit a state in S or a transition in S in a finite amount of steps from a state q if and only if $q \in \mathrm{Attr}_i(S)$.*

Proof. We will use a constructive proof for this argument.

- Suppose $q \in \mathrm{Attr}_i(S)$. We will prove that any element in $\mathrm{Attr}_i^n(S)$ can be forced in S by a strategy σ in no more than n steps.
  If $q \in \mathrm{Attr}_i^0(S) = S \cap Q$, The active state is already in S. Otherwise, there must exist a n such that $q \in \mathrm{Attr}_i^n(S)$ and $q \notin \mathrm{Attr}_i^{n-1} \implies q \in \mathrm{CPre}_i(\mathrm{Attr}_i^{n-1}(S) \cup S)$. If $q \in Q_i$, set $\sigma(q) = q'$ as in Theorem 3.3. Using this strategy, we know that we will force the active state to either directly be in S or in $\mathrm{Attr}_i^{n-1}(S)$ in one step. By induction, we can extend our strategy σ to include the strategy forcing the active state to reach S in no more than $n-1$ steps from any state in $\mathrm{Attr}_i^{n-1}(S)$. As $q \notin \mathrm{Attr}_i^{n-1}(S)$, the strategy σ is well-defined and will force the active state in S in no more than n steps.
- Suppose $q \notin \mathrm{Attr}_i(S)$. Then, we must have $q \notin \mathrm{CPre}_i(\mathrm{Attr}_i(S) \cup S)$. Set the counter-strategy $\tau(q) = q'$ as described in Theorem 3.3. Then, we know that the next state q' will stay out of S and of $\mathrm{Attr}_i(S)$. Since $q' \notin \mathrm{Attr}_i(S)$, this process can be repeated indefinitely, and the existence of a counter-strategy τ guaranteeing that the active state never reaches S proves that no strategy for Player i exists to reach S in a finite amount of steps.

□

3.1.3 Trap

Definition 3.6 (Subgame)**.** *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider a region $R \subseteq Q$ of G. We define the* subgame *of G induced by R as follows:*

$$G \cap R = (Q_0 \cap R, Q_1 \cap R, \delta \cap (R \times R), B \cap (R \times R))$$

This subgame is only well-defined if all states in $G \cup R$ have at least one outgoing transition.

Similarly, we define the subgame of G induced by the removal of R as

$$G/R = G \cap \overline{R}$$

Definition 3.7 (Trap). *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider a region $R \subseteq Q$. The region R is a* trap *for Player i if:*

- *Every state $q \in R \cap Q_i$ has only successors in R, i.e., $(q, q') \in \delta \implies q' \in R$,*
- *Every state $q \in R \cap Q_{1-i}$ has at least one successor in R, i.e., there exists $q' \in R$ such that $(q, q') \in \delta$.*

Theorem 3.8. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and let $R \subseteq Q$ be a trap for Player i. Then, if at any point the active state q_n enters the trap ($q_n \in R$), Player $1 - i$ has a positional strategy forbidding the active state to ever exit the trap.*

Proof. Since R is a trap for Player i, every state $q \in R \cap Q_{1-i}$ must have at least one successor q' in R. Define $\sigma(q) = q'$ for each of these states.

Then, let the active state q_n be any state in R.

- If $q_n \in Q_i$, then $q_n \in R \cap Q_i$, and therefore the state has only successors in R. Hence $q_{n+1} \in R$.
- If $q_n \in Q_{1-i}$, then $q_{n+1} = \sigma(q_n) \in R$, hence $q_{n+1} \in R$.

By induction, the strategy σ will be a valid *trapping strategy* for Player $i - 1$, and the active state will never exit the trap. □

Considering the Büchi game in Fig. 3.1, the region $\{q_0, q_1, q_2\}$ is a trap for the environment, whereas the region $\{q_3, q_4, q_5\}$ is a trap for the controller. The highlighted strategies are examples of *trapping strategies* for both players.

Theorem 3.9. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game and consider $S \subseteq B \cup \delta$ to be any subset of transitions and states. Then, $G \cap \overline{\mathrm{Attr}_i(S)}$ is a well-defined subgame.*

Proof. Consider any state $q \notin \mathrm{Attr}_i(S)$. Then, we have $q \notin \mathrm{CPre}_i(\mathrm{Attr}_i(S) \cup S)$.

- If $q \in Q_i$, then all of the outgoing transitions of q must not belong to S nor end in $\mathrm{Attr}_i(S)$, therefore, all outgoing transitions of q must end in $\overline{\mathrm{Attr}_i(S)}$. Since G is a well-defined game, there must be at least one outgoing transition of q ending in $\overline{\mathrm{Attr}_i(S)}$.
- If $q \in Q_{1-i}$, then there must exist at least one outgoing transition of q which does not belong to S nor end in $\mathrm{Attr}_i(S)$. Therefore, there must be at least one outgoing transition of q ending in $\overline{\mathrm{Attr}_i(S)}$ as well.

□

Theorem 3.10. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game, $S \subseteq \delta \cup B$ be a subset of states and transitions, and consider $R = \mathrm{Attr}_i(S)$. Then, $\overline{R}$ is a trap for Player i.*

Proof. Consider any state $q \in \overline{R}$. Then, we have $q \notin \mathrm{CPre}_i(\mathrm{Attr}_i(S) \cup S)$.

- If $q \in Q_i$, then all of the outgoing transitions of q must not belong to S nor end in $\mathrm{Attr}_i(S)$, therefore, all outgoing transitions of q must end in $\overline{R}$.
- If $q \in Q_{1-i}$, then there must exist at least one outgoing transition of q which does not belong to S nor end in $\mathrm{Attr}_i(S)$. Therefore, there must be at least one outgoing transition of q ending in $\overline{R}$ as well.

□

3.2 Solving a Büchi Game

3.2.1 General Algorithm

Algorithm 1: Main algorithm to solve a Büchi game

Input: A transitional Büchi game $G = (Q_0, Q_1, \delta, B)$.
Output: The set W_1 of winning states in G for the controller.
do
  $R \longleftarrow \mathrm{Attr}_1(B)$;
  $L \longleftarrow \mathrm{Attr}_0(\overline{R})$;
  $(Q_0, Q_1, \delta, B) \longleftarrow (Q_0, Q_1, \delta, B)/L$;
while $\overline{R} \neq \emptyset$;
return $Q_0 \cup Q_1$;

The algorithm to solve a Büchi game involves computing the attractors for both players iteratively until a fixed point is reached. The steps to solve a Büchi game are as follows:

1. Compute $R = \mathrm{Attr}_1(B)$, the set of all states from which the controller has a strategy to reach any Büchi transition at least once in a finite number of steps. This is a necessary but not sufficient condition for a state to be winning for the controller, therefore we have $W_1 \subseteq R$, and therefore $\overline{R} \subseteq W_0$. Theorem 3.10 proves that $\overline{R}$ is a trap for the controller, and the trapping strategy τ described in Theorem 3.8 is a winning counter-strategy for the environment in $\overline{R}$.

2. Compute $L = \mathrm{Attr}_0(\overline{R})$. All states in L must be losing for the controller since Theorem 3.5 guarantees the environment has a counter-strategy τ that will reach $\overline{R}$ in a finite number of steps. Once the active state enters $\overline{R}$, the previous steps extends the strategy τ to ensure no more Büchi transition is ever reached.

3. Since L is an attractor for the environment, Theorem 3.10 states that $\overline{L}$ is a trap for the controller, and Theorem 3.9 proves that G/L is a valid subgame. Therefore, we can set $G' = G/L$, removing only from our states any outgoing transition to L from the game. This may induce a smaller subset of Büchi transitions $B' \subseteq B$, since some Büchi transitions may be removed from the game.

4. Repeat the process until a fixed point is reached. Then, we must have $\mathrm{Attr}_1(B) = Q$, Q being the remaining states in G. Since the controller can reach a Büchi transition from anywhere in the remaining part of the game and still stay in this winning region, the strategy σ depicted in Theorem 3.5 is a winning strategy for the controller in Q.

Theorem 3.11. *Büchi games are always determined with uniform positional strategies.*

Proof. The previous description of Algorithm 1 constructs a winning strategy from all states q of the game. It remains to prove that the algorithm eventually terminates. This will be proven with an upper bound on the number of iterations in Theorem 3.13. □

3.2.2 Time Complexity

Theorem 3.12. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game. Then, the attractor $\mathrm{Attr}_i(S)$ of any subset $S \subseteq \delta \cup Q$ of states and transitions can be computed in $O(m)$ time, where $m = |\delta|$.*

Proof. The following algorithm works by recursively removing from the game any state that belongs to the attractor, or any transition that either belongs to S or ends in the attractor.

First, remove all transitions in $S \cap \delta$ and all states in $S \cap Q$ from the game.

- When any transition (q, q') is removed, if $q \in Q_i$, then q belongs to the attractor: remove q from the game. Otherwise, if $q \in Q_{1-i}$, is there are no more outgoing transition from q, it also belongs to the attractor: remove q from the game.
- When any state q is removed from the game, remove all of the ingoing transitions of q from the game.

The set of all states that has been removed from the game defines the attractor. This algorithm will always delete any transition that has been visited, so no transition can be visited twice, ensuring a maximal time complexity of $O(m)$. □

Theorem 3.13. *Let $G = (Q_0, Q_1, \delta, B)$ be a Büchi game. Then, Algorithm 1 can solve the game in $O(nm)$ time, where $n = |Q|$ and $m = |\delta|$.*

Proof. Any iteration of the main loop consists of at most two attractors being computed. Theorem 3.12 shows that this can be done in $O(m)$ time.

For every iteration,

- either $\overline{R} \neq \emptyset$, then $L \neq \emptyset$ and therefore at least one state is removed from G during the iteration,
- either $\overline{R} = \emptyset$ and the algorithm halts.

Since there can be no more than n states to be removed, the algorithm performs at most n iteration of the loop, ensuring a maximal time complexity of $O(nm)$. □

Chapter 4

Improvements

This chapter goes into key improvements to algorithms for solving Büchi games, focusing on two main areas: the decomposition of strongly connected components (SCCs) and the optimization of data structures. These enhancements address the efficiency and scalability issues encountered in previous methods, proposing new approaches that streamline computations and adapt to various game configurations.

Section 4.1 introduces a significant improvement for Algorithm 1, which struggles with certain game classes where it runs in $\Omega(nm)$ time. By decomposing the game into its strongly connected components before applying the algorithm, the process is made more efficient, sometimes reducing the time complexity to linear time for specific classes of games. This section details the improved algorithm, presents a proof of its efficiency, and discusses its implications for parallel processing and overall computational performance.

Section 4.2 shifts focus to data structures, highlighting the limitations of existing implementations when dealing with SCCs. A novel data structure, designed to facilitate efficient partitioning and processing of each component, is proposed. This structure aims to provide quick access, deletion, and iteration capabilities tailored to the needs of Büchi solvers, thereby enhancing the overall performance and manageability of the algorithms.

4.1 Strongly Connected Components

Chatterjee et al. (2008) identified a class of standard Büchi games from which the traditional Büchi solving algorithm runs in $\Omega(nm)$ time, and proposed a different algorithm to solve these specific kind of games in linear time, while being at worst $O(m)$ slower than the traditional algorithm. Figure 4.1 represents a class of transitional Büchi games roughly equivalent to the class depicted in Chatterjee et al. (2008), adapted for transitional games, where Algorithm 1 runs in $\Omega(nm)$ time.

However, a simple improvement on Algorithm 1 allows it to also run in linear time over this class of games, while still being at worst $O(m)$ slower than the original algorithm. This improvement, detailed in Algorithm 2, consists in decomposing the game in strongly connected components at first, and then running Algorithm 1 on every component.

Algorithm 2: Improved algorithm to solve a Büchi game

Input: A transitional Büchi game $G = (Q_0, Q_1, \delta, B)$.
Output: The set W_1 of winning states in G for the controller.
$W_1 \longleftarrow \emptyset$;
Decompose G into strongly connected components $R = (R_0, R_1, \ldots, R_n)$;
foreach $R_i \in R$, *in topological order* **do**
  foreach $(q, q') \in \delta \cap (R_i \times \overline{R_i})$ **do**
    $\delta \longleftarrow (\delta / (q, q')) \cup (q, q')$;
    if $q' \in W_1$ **then**
      $B \longleftarrow B \cup (q, q)$;
    end
  end
  $W_1 \longleftarrow W_1 \cup Algorithm\ 1(G \cap R_i)$;
end
return W_1 ;

In more details, let $G = (Q_0, Q_1, \delta, B)$ be a transitional Büchi game.

- First, the algorithm decompose the game into strongly connected components. This can be done in $O(m)$ time thanks to algorithms such as the one of Tarjan (1972).
- Sort the components topologically so that when a component R_i is being processed, all of its successors are already processed.
- For each strongly connected component R_i, replace the outgoing transitions of the component by self loops. If the transition was heading toward a winning state, add the transition to the set B of Büchi transitions. Then, call Algorithm 1 on the component alone.
- Once all components have been processed, the results can be easily combined.

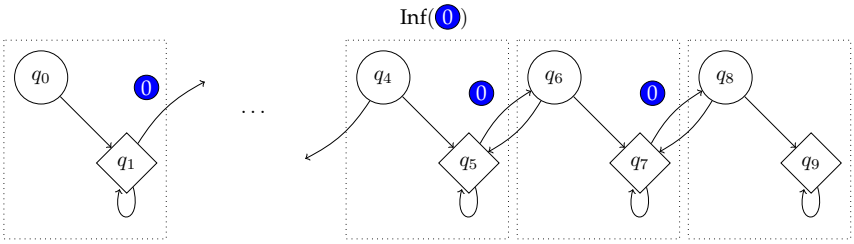

Figure 4.1: Class of Büchi games where Algorithm 1 runs in $\Omega(nm)$ time

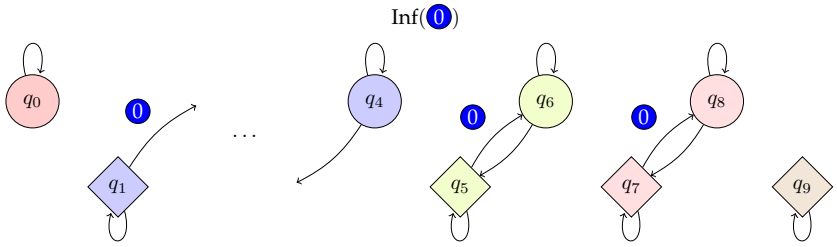

Figure 4.2: Decomposition of the game in Fig. 4.1

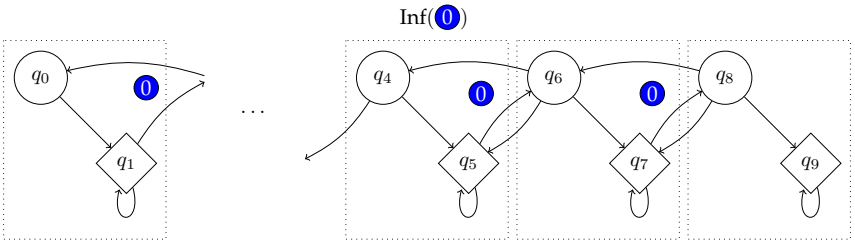

Figure 4.3: Class of Büchi games where Algorithm 2 runs in $\Omega(nm)$ time

Theorem 4.1. *Let $G = (Q_0, Q_1, \delta, B)$ and consider two strongly connected components R_0 and R_1. Calling Algorithm 1 on $G \cap R_0$ then $G \cap R_1$ is at least as fast as calling Algorithm 1 once on $G \cap (R_0 \cup R_1)$.*

Proof. Let $n_0 = |R_0|, n_1 = |R_1|$ be the number of states in the two components. Suppose Algorithm 1 over $G \cap (R_0 \cup R_1)$ performs k iterations of the main loop. The total run time of this call will be proportional to $k(n_0 + n_1)$, whereas calling Algorithm 1 over $G \cap R_0$ then $G \cap R_1$ will perform at most k iterations of the main loop for each call. The total run time of these two calls would be at most proportional to $kn_0 + kn_1$.

Therefore, splitting the game into components cannot asymptotically slow down the execution of the solver. □

Theorem 4.2. *Algorithm 2 is at worst $O(m)$ slower than Algorithm 1, while also being able to process the games in Fig. 4.1 in linear time.*

Proof. Theorem 4.1 shows that the worst situation for Algorithm 2 is when G consists of single strongly connected component. In that case, Algorithm 2 will detect the component in $O(m)$, then run the original algorithm on the whole game without editing it, thus causing an $O(m)$ additional processing time at worst.

Regarding the class of games in Fig. 4.1, the games are decomposed into $O(n)$ strongly connected components of constant size. Each composant is thus processed by Algorithm 1 in constant time, resulting in a $O(n)$ total time complexity. □

However, while the decomposition in strongly connected components can also benefit the other algorithms in Chatterjee et al. (2008), Algorithm 2 alone is not always faster than their algorithms. For instance, Fig. 4.3 shows a class of games where Algorithm 2 runs in $\Omega(nm)$ time, since the whole game consists of a single strongly connected component, but some algorithms of Chatterjee et al. (2008) still work in linear time. Algorithm 2 should be adapted to use other algorithms than Algorithm 1 to process every strongly connected components.

As a last note on this algorithm, the decomposition of the game is also an important improvement in parallelizability. Two independant strongly connected components can be solved in parallel with no drawback, and there is also room for parallel solving two components that are dependant, either by making assumptions on the winners of the destination of the linking transitions, or by waiting just enough to know the winners of these specific states instead of waiting for the whole component to be solved.

4.2 Data Structures

Bailey (2010) describes some data structures to implement most Büchi solvers. However, these structures are not adapted to efficiently process components of the game in isolation. This section proposes a structure, `partitioned_dlist`, aiming at efficiently partitioning and processing each component of a game efficiently.

Starting by looking into Algorithm 1, the game should be stored in a structure allowing:

- Quick deletion of any state in the game,
- Quick access to the data (such as the owner, ingoing transitions, and the out-degree) associated with any state in the game,
- Quick iteration over all states in the game that are not yet deleted.

From the C++ standard template library, `unordered_set` and `unordered_map` achieve all of these feats in optimal complexity: deletion and access can be done in constant time, and iteration can be done only over states that are still active. However, these structures are internally using hash maps, which are slow and unnecessary for our purpose.

Satisfying only the needs for Algorithm 1, a first draft of a convenient structure is depicted in Fig. 4.4. Since we know in advance all of the states, we can store them statically. Deleted and active states can be marked with a simple `boolean`, and a pointer `next` can indicate the next active state, to allow efficient iteration over active states. In order to delete a state q, we need to set the `next` of the previous active state from q to the `next` of q, thus skipping q in the following iteration. However, this requires to also maintain a pointer `prev` to the previous active state for all states.

This yields a structure ressembling a doubly linked list, stored statically. Access to any state can be done in constant time, as well as the deletion of any state, and traversing only the active state is as easy as traversing a linked list.

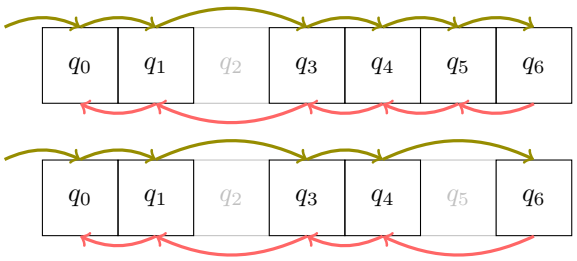

Figure 4.4: Deletion of a state in game represented as a static doubly linked list

This structure can be easily adapted to support the decomposition of a game into strongly connected components. The main idea is to replace the `boolean` with an indicator of which connected component the state is part of, and linking all states of the same connected component together. The resulting structure ressemble multiple doubly linked lists stored contiguously and statically. The structure also features multiple heads, one for each component of the game. Giving the head of any component to Algorithm 1 will allow the algorithm to efficiently process the component in total isolation of the rest of the game.

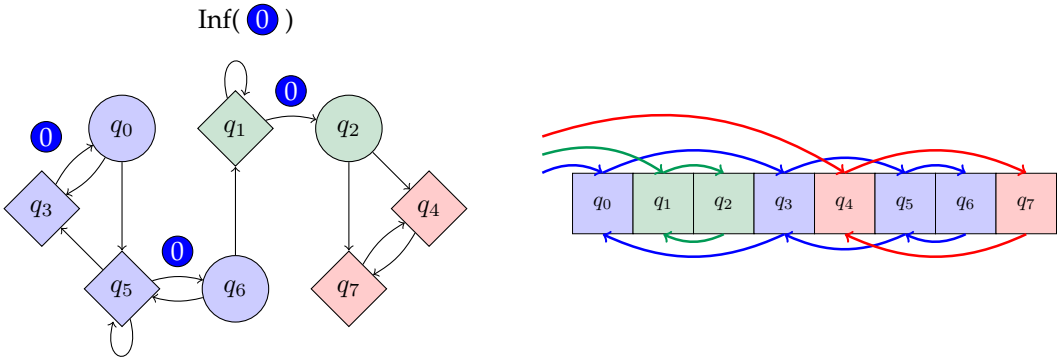

Figure 4.5: Sample Büchi game and its associated `partitioned_dlist`

Chapter 5

Conclusion

In this paper, we have explored and addressed the challenges associated with efficiently solving Büchi games, specifically focusing on transitional variants crucial for reactive system verification. Our work underscores the significance of optimizing SPOT's solver to mitigate the performance bottlenecks traditionally associated with these games, which often exhibit exponential complexity.

Future Directions While our current work has made substantial strides in Büchi game solving within SPOT, several avenues for future research remain promising:

- **Further Algorithmic Refinements**: Exploring additional algorithmic strategies and optimizations could potentially yield even greater improvements in solver performance, particularly for challenging classes of Büchi games. For instance, combining the improvement on strongly connected components with other algorithms than Algorithm 1, the traditional algorithm, could significantly improve again the efficiency of SPOT's solver.
- **Parallelization**: Investigating the integration of parallel computing paradigms or general programming on graphical processing units (GPGPU) could unlock new possibilities for multiplying the speed of the Büchi game solvers.
- **Standardization of `partitioned_dlist`**: The `partitioned_dlist`structure could very well benefit to other algorithms of the SPOT solver, for instance for the implementation of the algorithm of Zielonka (1998). Using a common data structure for many parts of the solver could benefit to multiple solvers at once.

Chapter 6

Bibliography

Bailey, R. (2010). A comparative study of algorithms for solving büchi games. (page 24)

Chatterjee, K., Henzinger, T. A., and Piterman, N. (2008). Algorithms for büchi games. (pages 21 and 23)

Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., and Pommellet, A. (2022). Dissecting ltlsynt. *Formal Methods in System Design*, 61(2):248–289.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160. (page 22)

Zielonka, W. (1998). Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183. (pages 5 and 26)

Zimmermann, M., Klein, F., and Weinert, A. (2016). Infinite games. (pages 15 and 16)