BUFFERED SIMULATION FOR BÜCHI AUTOMATA

Dissertation zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

im Fachbereich Elektrotechnik/Informatik der Universität Kassel

Milka Hutagalung

Mai 2018

Erster Gutachter:	Prof. Dr. Martin Lange
Zweiter Gutachter:	Prof. Dr. Antonín Kučera
Tag der Disputation:	27. September 2018

Abstract

We introduce a new family of simulation relations between two non-deterministic Büchi automata (NBA) called buffered simulation. We extend the game framework of the standard fair simulation such that Duplicator can skip her turn and store the letter that is read by Spoiler temporarily to a buffer. Duplicator then can execute the letters in the buffer in some later round. In this way, she has a preview of Spoiler's move and has more chances to mimic Spoiler's run correctly than in the standard fair simulation game. We generalise such a simulation to the case where multiple buffers are involved. In such a case, a rule that tells us to which buffers a letter should be stored is given. Once Spoiler reads a letter, Duplicator stores it to all the associated buffers, and when she wants to execute the letter, she has to pop it from all the associated buffers.

We study the decidability and complexity of buffered simulation with one or multiple buffers. We show that buffered simulation is undecidable if some of the buffers are unbounded. It is even already highly undecidable in the case of two buffers where one is unbounded and the other one is of capacity 0. In the case where all buffers are bounded, buffered simulation is decidable in polynomial time. In the case of a single, but unbounded buffer, buffered simulation is PSPACE-complete for a variant that requires Duplicator to pop all the letters from the buffer each time she decides to move. It is, however, EXPTIME-complete for the general case.

We further show that buffered simulation with one buffer can be used to incrementally approximate language inclusion between two NBA. In the case of multiple buffers, it can be used to approximate a more general problem, namely the trace closure inclusion problem, which is known to be highly undecidable. We give a theoretical justification of buffered simulation by giving a characterisation using the notion of continuity. Buffered simulation with unbounded buffers can be characterised by the existence of a continuous function that witnesses the language or trace closure inclusion between the input automata. We can lift such a characterisation to the case of bounded buffers by considering a Lipschitz continuous function instead of just a continuous one. Such a characterisation, however, only holds for some restricted automata called cyclic-path-connected automata.

Zusammenfassung

Wir stellen eine neue Familie von Simulationsrelationen zwischen zwei nicht deterministischen Büchi Automaten (NBA), genannt gepufferte Simulation vor. Wir erweitern das Spiel-Framework der üblichen fairen Simulation, so dass DUPLICATOR ihren Spielzug überspringen und die Buchstaben, die SPOILER gelesenen hat, vorübergehend in einem Puffer speichern kann. DUPLICATOR kann diese Buchstaben in ihrer Struktur dann später ausführen. DUPLICATOR hat damit eine Vorschau in die Bewegungen von SPOILER und somit mehr Chancen, den Lauf SPOILERS korrekt nachzuahmen als im Standard-Simulationsspiel. Wir verallgemeinern dann diese Simulationsrelationen auf mehrere Puffer. In einem solchen Fall gibt es eine Regel, die angibt, in welchen Puffern ein Buchstabe gespeichert werden soll. Sobald SPOILER einen Buchstaben liest, speichert DUPLICATOR ihn in jedem zugeordneten Puffer und wenn DUPLICATOR ihn in ihrer Struktur ausführen möchte, muss sie den Buchstaben von jedem zugehörigen Puffer löschen.

Wir untersuchen die Entscheidbarkeit und Komplexität von gepufferten Simulationen mit einem oder mehreren Puffern. Wir zeigen, dass gepufferte Simulation nicht mehr entscheidbar ist, wenn einige Puffer unbegrenzt sind. Es ist tatsächlich sogar schon hochgradig unentscheidbar bei zwei Puffern, in denen einer unbegrenzt ist und der andere die Kapazität 0 hat. In dem Fall, in dem alle Puffer begrenzt sind, ist gepufferte Simulation in polynomieller Zeit entscheidbar. Für die Simulation mit einem einzelnen, aber unbeschränkten Puffer ist die gepufferte Simulation PSPACE-vollständig für eine Variante, bei der DUPLICATOR alle Buchstaben aus dem Puffer löschen muss, jedes Mal wenn sie sich bewegt. Es ist jedoch EXPTIME-vollständig für den allgemeinen Fall.

Wir zeigen weiterhin, dass gepufferte Simulation mit einem Puffer Spracheinklusion zwischen zwei NBA schrittweise approximiert. Mit mehreren Puffern kann gepufferte Simulation die Inklusion des Spurabschlusses zweier NBA approximieren, welche bekanntermaßen hochgradig unentscheidbar ist. Wir formulieren eine theoretische Untermauerung für gepufferte Simulation, indem wir sie als Stetigkeitsfrage in einem geeigneten topologischen Raum auffassen. Gepufferte Simulation mit unbeschränkten Puffern kann durch die Existenz einer stetigen Funktion, die die Sprache oder die Spurabschluss Inklusion zwischen den beiden Automaten bezeugt, charakterisiert werden. Wir können diese Charakterisierung auf gepufferte Simulation mit begrenzten Puffern erweitern, indem wir sogar Lipschitz-Stetigkeit der Funktion verlangen. Eine solche Charakterisierung gilt allerdings nur für eine beschränkte Klassen von Automaten, die wir zyklisch-pfad-verbunden nennen.

Acknowledgements

First of all, I would like to express my deepest gratitude to my advisor, Prof. Dr. Martin Lange, for the opportunity to join his research group and for the academic guidance he has given me throughout my doctoral work. The discussion with him and his continuous support have encouraged me to explore various research questions and approaches. They have enormously helped me produce this dissertation. I would also like to thank him for his tremendous support, especially after I had my first son some years ago. Without his support, I would never have a flexible time to complete this dissertation.

I would like to extend my gratitude to Prof. Dr. Étienne Lozes for his advice in the initial stage of my research. I am very grateful for his comments regarding the decidability of buffered simulation with one unbounded buffer. They have given me an invaluable insight. I also thank Prof. Dr. Antonín Kučera for being the second reviewer of this dissertation and Prof. Dr. Dietrich Kuske for his practical suggestions that have enriched this work considerably.

I have had the great pleasure of working with all my former colleagues in the University of Kassel. Without them, my lunchtime in the university would have been very plain and boring. I particularly would like to thank Florian Bruse for proofreading the draft of this dissertation and pointing out some bug in one important lemma I overlooked. I wish to thank Daniel Kernberger for giving me a useful feedback for the last rehearsal of my thesis defense and Norbert Hundeshagen for many helpful tips and advice on finishing the doctoral work. I also owe many thanks to Michael Möller, Tina Landefeld and Lara Yörük for their practical help during my time in the university.

This dissertation would not have been possible without the support of my parents, son and siblings. They are the source of my strength. I am extremely grateful for their unwavering support during my tough times. I would have given up many times had it not been for their support.

Last but not least, a very special thank to my husband, David, for his endless love and support. I especially thank him for encouraging me to pursue these doctoral studies and for always listening to my countless doubts and worries. Words cannot express how grateful I am to have his support through the ups and downs of writing this dissertation.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlichen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren durch mich verwendet worden.

Milka Hutagalung

Contents

List of Figures

Chapter 1

Introduction

Automata. Automata play an important role in computer science. They are known to have many applications in diverse area such as in compiler design for parsing and lexical analysis [ASU86, HMU06], in XML processing for document validation and query processing [Sch07, Koc09], in system verification and static program analysis for modelling the system and properties that want to be verified [VW86, NNH99], in classical logic to obtain a decision procedure for monadic-second-order logic [Büc62], in natural language processing to represent natural language [KK94, LK93], in DNA sequence analysis to model the evolution and role of DNA sequences [BF84, TPKC07], and many more. This is because automata are one of the most basic models of computation. We can model systems or programs as automata and then analyse their computation and behaviour by looking at the automata model.

For systems or programs that always terminate, such as parsers, search engines, or compilers, one uses the notion of finite automata. However nowadays there are also many systems that no longer transform an input to output and then terminate. Instead, they continuously interact with the user or environment. Some examples of such systems are web servers, communication protocols, or operating systems. For such non-terminating reactive systems, one often uses the notion of ω-automata.

One of the simplest forms of ω-automata are Büchi automata. They are widely used to model reactive systems and to specify their non-terminating behaviour [EH00, Hol04]. Intuitively, a Büchi automaton can be seen as a directed graph where the edges are labeled, and some nodes are considered to be final or accepting. The nodes represent the states of the system and an a-labeled edge represents the state-changing of the system when the action a is executed. A Büchi automaton runs on a given input word that has an infinite length. The run goes through an infinite sequence of states and the input word is considered to be valid or accepted if the run visits a final state infinitely often. A Büchi automaton can be seen as an acceptor of a language over infinite words. For example, the Büchi automaton $\mathcal{A}$ that is depicted in Figure 1.1 accepts the language $L(\mathcal{A})$ that consists of two infinite words: aba^ω and aca^ω.

Language Inclusion. By considering the automaton-model, we can reduce many problems regarding the analysis of system design to the problems over automata, which then can be solved using some algorithms on graph. For example, one of the most important problems in computer science is the problem of checking whether a system behaves accordingly with respect to a given specification, e.g. never reaches a deadlock, is logically correct, etc. Such a problem becomes undoubtedly important especially in a life-critical

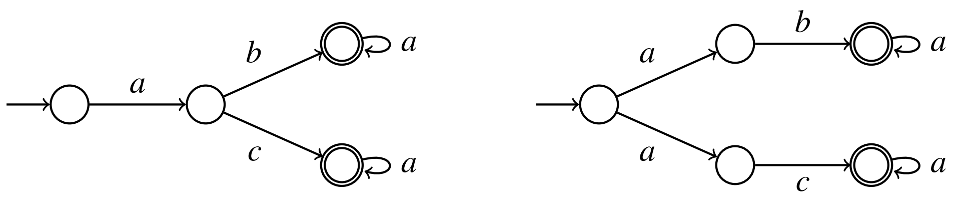


Figure 1.1: A pair of Büchi automata $\mathcal{A}$ (left) and $\mathcal{B}$ (right) in which $\mathcal{A} \not\sqsubseteq \mathcal{B}$.

system where some occurrence of errors can have an extremely expensive cost to repair or even threaten our life. For example, in programs that control part of medical system, or in flight control programs. For such reactive non-terminating systems, testing is not enough and one needs to formally verify the systems.

The problem of verifying whether such systems meet the desired specification can be reduced to the problem of deciding language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ between two Büchi automata $\mathcal{A}$ and $\mathcal{B}$. The automaton $\mathcal{A}$ represents the computation of the system and the automaton $\mathcal{B}$ represents the desired computation given by the specification. Unfortunately, the problem of deciding language inclusion between two Büchi automata is computationally hard. It is PSPACE-complete [MS73]. Hence one of the major challenges in the field of automata theory is to find a good and decent approximation for language inclusion.

Simulation. Simulation is a preorder relation between the states of an automaton. We say that a state p simulates a state q if p can mimic all stepwise behaviours of q. We can use simulation to approximate language inclusion between two Büchi automata $\mathcal{A}$, $\mathcal{B}$, by looking at how their initial states relate [DHWT92]. If the initial state of $\mathcal{B}$ simulates the initial state of $\mathcal{A}$ then we can conclude that $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

There is a game characterisation for simulation. The game is played by two players called Spoiler (male) and Duplicator (female). Initially, two pebbles are placed each in the initial states of $\mathcal{A}$ and $\mathcal{B}$. In each round, Spoiler moves the pebble in $\mathcal{A}$ one step by reading a letter and Duplicator tries to mimic this move by moving the pebble in $\mathcal{B}$ one step by reading the same letter. This continues round by round. If one of the players eventually gets stuck then the opponent wins, otherwise Duplicator wins if whenever Spoiler visits a final state infinitely often, she also visits a final state infinitely often. If Duplicator wins no matter how Spoiler plays then we say that the automaton $\mathcal{B}$ simulates $\mathcal{A}$, i.e. $\mathcal{A} \sqsubseteq \mathcal{B}$, and hence $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds.

For any two Büchi automata $\mathcal{A}$, $\mathcal{B}$, unlike language inclusion, deciding whether Duplicator wins the simulation game can be done in polynomial time. This, however, comes at price. There are many pairs of automata where simulation fails to show language inclusion. For example, consider the two Büchi automata $\mathcal{A}$, $\mathcal{B}$ depicted in Figure 1.1. This is a standard example where simulation fails to show language inclusion. First note that $L(\mathcal{A}) \subseteq L(\mathcal{B})$. The languages of $\mathcal{A}$ and $\mathcal{B}$ are indeed the same. Duplicator, however, loses the simulation game. In the first round, after Spoiler reads a, Duplicator has to move the pebble to one of the successors of the initial state of $\mathcal{B}$. Without loss of generality, suppose to the one with a b-transition. In the next round, Spoiler can choose to read c and make Duplicator get stuck. Hence Duplicator loses the simulation game.

The reason why simulation fails to show language inclusion is because Duplicator is too weak to guess the run that will be formed by Spoiler. Duplicator might move the pebble to some successor which prevents him to form a corresponding run in the future. In the literature, there are many attempts that try to extend simulation by adding more power to Duplicator or by restricting the move of Spoiler such that simulation gets closer to language inclusion.

- In the *static k-letter simulation* that is introduced in [HLL13], instead of only one step, Spoiler is forced to move the pebble $k > 0$ steps in each round. Duplicator then has more information about Spoiler's run in each round. For example, consider again the two automata $\mathcal{A}$, $\mathcal{B}$ from Figure 1.1. Duplicator wins the static 2-letter simulation game. In the first round, Spoiler is forced to read either ab or ac. Duplicator then can mimic this move by going to one of the accepting states in $\mathcal{B}$ by reading the same word. From the accepting state, Duplicator can mimic any of Spoiler's move and win the game. By considering such an extended simulation, we can show language inclusion in cases which cannot be shown by the standard simulation.

- In the *k-pebble simulation*, the standard simulation game is extended such that Duplicator has more power [Ete02]. She can control $k > 0$ pebbles, and hence has more chances to correctly mimic Spoiler's move. For example, if we consider again the two automata $\mathcal{A}$, $\mathcal{B}$ given by Figure 1.1 then Duplicator wins the 2-pebble simulation game. In the first round, after Spoiler reads a, Duplicator moves her two pebbles each to the a-successors of the initial state of $\mathcal{A}$. In the second round, after Spoiler reads b or c, Duplicator moves one of her pebbles to reach the final state and drops the other one. From this state, Duplicator can play accordingly and show language inclusion.

- In the *k-lookahead* and *dynamic k-letter simulations* that are introduced independently in [CM13] and in [HLL13], in each round i, Duplicator chooses $\ell_i \leq k$ and Spoiler is forced to move ℓ_i steps. Such a simulation indeed extends the static k-letter simulation. The static k-letter simulation game is a special case of the dynamic one where in each round i, Duplicator always chooses $\ell_i = k$. It is shown that the dynamic k-letter simulation game has more advantages. It can show more language inclusions than the static one and admits a hierarchy, i.e. whenever Duplicator wins the k-lookahead or dynamic k-letter simulation game, she also wins the one with parameter $k' > k$. This property does not hold for the static k-letter simulation.

- In the *delay simulation* that is introduced in [HKT10], the extended simulation game is played on a language L of pairs of infinite words with respect to a delay function $f : \mathbb{N}^+ \to \mathbb{N}^+$. In every round $i > 0$, Spoiler chooses a word of length $f(i)$ and Duplicator chooses a letter. The play goes on for infinitely many rounds and Duplicator wins iff the pair of infinite words formed by Spoiler and Duplicator belongs to L. Such a simulation generalises the one over automata. For example, we can see the standard simulation between two Büchi automata $\mathcal{A}$, $\mathcal{B}$ as a special case of delay simulation in which L consists of pairs of accepting runs of $\mathcal{A}$, $\mathcal{B}$ that are over the same word and $f(i) = 1$ for all $i > 0$.

Buffered Simulation. In this work, we extend the underlying idea of the k-lookahead or the dynamic k-letter simulations. We extend the game framework of the standard simulation with a FIFO buffer such that Duplicator can skip her turn and use the buffer to

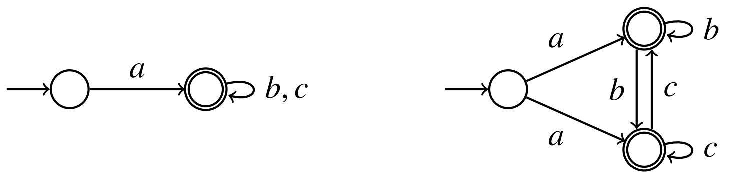


Figure 1.2: A pair of Büchi automata $\mathcal{A}$ and $\mathcal{B}$.

temporarily store the letter that is read by Spoiler. Duplicator can pop some or all the letters from the buffer in some later round to continue her move. The buffer can have a bounded or an unbounded capacity.

For example, consider the automata $\mathcal{A}$, $\mathcal{B}$ in Figure 1.2. In this case, Duplicator loses the k-lookahead or the dynamic k-letter simulation for any $k \in \mathbb{N}$. At the end of the first round, Spoiler's pebble is in the accepting state of $\mathcal{A}$ and Duplicator's pebble is in one of the accepting states of $\mathcal{B}$. Let us assume that Duplicator's pebble is in the accepting state with b-transition. In the second round, Spoiler can make Duplicator get stuck by reading c. Hence Duplicator loses k-lookahead or dynamic k-letter simulation games for any $k > 0$. Duplicator, however, wins the simulation game with a buffer of size 1. In the first round, after Spoiler reads a, Duplicator skips her turn and puts a to the buffer. In the second round, after Spoiler reads b or c, Duplicator pushes it to the buffer. She then pops a and moves the pebble to the accepting state with b-transition if Spoiler reads b, and to the one with c-transition if Spoiler reads c. In this way, she will not get stuck in the next round. Duplicator does this similarly for the rest of the play, i.e. she always keeps her run one step behind Spoiler's run. She uses the buffer to store the last letter that is read by Spoiler. Duplicator wins the game with a buffer of capacity 1 and shows language inclusion between $\mathcal{A}$, $\mathcal{B}$ which cannot be shown with the lookahead or dynamic multi-letter simulations.

In the field of formal languages, there is a theory that extends the concept of words and languages to model concurrent computations. Recall that in concurrent computations, some actions or processes can be executed simultaneously. The actions or processes that are executed simultaneously are considered to be independent of each other. The theory of *Mazurkiewicz traces* has been introduced to model such computations [Maz89, DR95]. Mazurkiewicz traces or just traces basically extend the concept of words in which some letters are allowed to commute. The letters that can commute model the processes that are independent of each other, and the letters that cannot commute model the ones that are dependent on each other. Intuitively, if two processes are independent of each other, the order of which process should be executed first or later does not matter. For example, if we consider traces over the letters a, b, c in which a, b are independent of each other, but not with c then the word $caba^\omega$ is considered to be trace equivalent to the word $cbaa^\omega$. This is because the second and the third letters, i.e. a and b, is allowed to commute with each other. In fact, the letter b is allowed to commute with any a in the word $caba^\omega$. Hence $caba^\omega$ is also trace equivalent to any word of the form ca^*ba^ω.

For any language L, the trace closure of L is denoted with $[L]$. It consists of any words that are trace equivalent to some word in L. For example, consider the automaton $\mathcal{A}$ in Figure 1.3 and suppose the letters a, b are independent of each other, but not with c. The language of $\mathcal{A}$ then is $L(\mathcal{A}) = \{caba^\omega\}$ and its trace closure is $[L(\mathcal{A})] = ca^*ba^\omega$. The trace closure $[L(\mathcal{A})]$ intuitively represents any possible sequential computations that can be executed by the concurrent system modeled by $\mathcal{A}$. Given two automata $\mathcal{A}$, $\mathcal{B}$, the

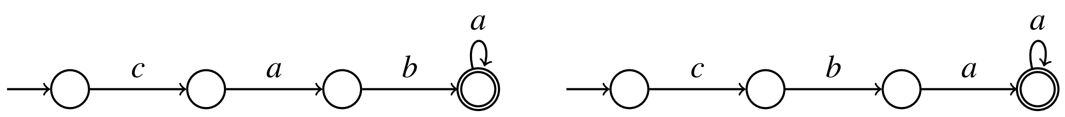


Figure 1.3: A pair of Büchi automata $\mathcal{A}$ and $\mathcal{B}$.

problem of checking whether $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$ then models the verification of systems that admit concurrency [Sak92, MSB+16]. The system and the specification that we want to check are modeled respectively by the automata $\mathcal{A}$ and $\mathcal{B}$. Unfortunately, checking trace closure inclusion between two Büchi automata is known to be highly undecidable [Sak92, Fin12].

We can slightly extend the framework of buffered simulation to approximate the trace closure inclusion. Consider a game where instead of only one FIFO buffer, Duplicator can use multiple FIFO buffers to temporarily store Spoiler's letters. Unlike in the case where only one buffer is involved, in this case, we assume that there is a rule that associates each letter to one or several buffers. Once Spoiler reads a letter, a copy of this letter gets pushed into all of the associated buffers. However if Duplicator would like to read the letter, she also has to pop it from all the associated buffers. Such a simulation can be used to approximate trace closure inclusion. Intuitively, we model the independence between letters via the rule that associates letters to buffers. A pair of independent letters should not share a buffer and a pair of dependent letters should share at least one buffer. For example, consider the two Büchi automata $\mathcal{A}$, $\mathcal{B}$ as in Figure 1.3 in which the letters a, b are independent of each other, but not with c. In this case, we consider a game with two buffers in which the first one is associated with a, c and the second one with b, c.

In the first round, after Spoiler reads c and pushes a copy of c to the first and second buffers, Duplicator reads c and pops them immediately from the first and the second buffers. In the second round, Spoiler will read a and pushes a copy of it to the first buffer. Duplicator cannot do anything except to skip her turn and let a go to the first buffer. In the third round, after Spoiler reads b and pushes it to the second buffer, Duplicator pops b then a and reads ba. She then reaches the accepting state from which she can simulate any of Spoiler's move and win the game. Thus Spoiler produces the word $caba^\omega$ and Duplicator produces cba^ω, a word that is trace equivalent to Spoiler's word. In general, if Duplicator wins the corresponding buffered simulation game then we have the inclusion $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$ which is equivalent to the trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$.

Despite of its simple and natural definition, the decidability and complexity of buffered simulation however is not trivial. This is true even in the case where we only have one buffer with bounded capacity. In the case of static, dynamic, and lookahead simulation games, the problem of deciding whether Duplicator wins can be done in polynomial time for a fixed parameter k. However, we cannot simply use this result for buffered simulation since we have seen that simulation with one buffer of capacity 1 is even more expressive than any k-lookahead or dynamic k-letter simulation. In the case where we consider an unbounded buffer, the decidability and complexity of buffered simulation then becomes even harder. Many questions regarding systems with an unbounded FIFO buffer are often undecidable [CF05]. Decidability of buffered simulation with an unbounded buffer may therefore be seen as surprising. The decidability question also gets more non-trivial in the case of multiple buffers. It is not obvious whether deciding buffered simulation with multiple buffers is still possible. If yes then we have a decidable approximation for a highly undecidable problem. However, if not then it would also be interesting to know how undecidable it is and whether there are some decidable instances.

It is interesting to know how good buffered simulation is to approximate trace closure or language inclusion, i.e. whether there are pairs of automata $\mathcal{A}$, $\mathcal{B}$ in which buffered simulation fails to show their inclusion. If there are such pairs, it is also interesting to know whether we can classify such automata and find their characteristics.

Thesis Outline. This thesis studies buffered simulation together with its decidability and complexity. We will consider both of the cases where only one buffer is involved and where multiple buffers are involved. The first one is basically a natural extension of the standard simulation which approximates language inclusion. The second one approximates a more general problem than language inclusion namely the trace closure inclusion. This thesis is presented as follows.

In Chapter 2, we recall the basic notions regarding words, languages, automata, decision problems, games, and simulation. We use the standard definitions and notations as in the literature. We also list some extended simulation that are available in the literature. We show their basic definitions and recall their important property.

In Chapter 3, we give the formal definition of buffered simulation. We first show the framework of buffered simulation with one buffer and then consider its natural extension to the case where multiple buffers are involved. We present the expressive power of buffered simulation and consider some of its restricted variants called the flushing variants. We compare simulation with one buffer to the extended simulations that are mentioned in Chapter 2.

Chapter 4 is the main part of this work. In this chapter, we give the complexity results regarding buffered simulation. We start with the case where all buffers are bounded. We then continue to the case where some buffers are unbounded. If there is only a single buffer and the capacity is unbounded, solving buffered simulation is still decidable. The complexity even gets better when we consider its flushing variant. However, in the case where multiple buffers are involved and one of them is unbounded, buffered simulation is undecidable. We then show how undecidable such a problem is.

In Chapter 5, we give the application of buffered simulation in the field of formal languages. Buffered simulation and its flushing variant can be used to incrementally approximate language and trace closure inclusion. We further show a characterisation of buffered simulation. Buffered simulation, in general, can be characterised by the existence of a continuous function that witnesses the trace closure inclusion between the given two automata $\mathcal{A}$, $\mathcal{B}$. This allows us to classify pairs of Büchi automata in which their language or trace closure inclusion cannot be shown with buffered simulation. We can even refine the characterisation of buffered simulation to the case of bounded buffers by considering a Lipschitz continuous function. However, such a refined characterisation only holds for some restricted class of automata called cyclic-path-connected automata.

In Chapter 6, we give the conclusion of our work. We list some open problems regarding buffered simulation for further works.

Publications Parts of this thesis have been published in some refereed proceedings. A slightly different version of Section 3.1 and Section 3.2 can be found in [HLL13] and [HLL14]. Some preliminary works of Section 3.4 and Section 3.5 have appeared in [HHK$^+$16a] and [HHK$^+$16b]. Section 4.3 and Section 4.4 have appeared in [HHK$^+$18] and Section 5.3 has been published in [Hut17].

Chapter 2

Preliminaries

We recall the basic notions that will be used throughout this work. We will start by considering *formal languages* and *decision problems*. We then continue with the notion of *automata* and consider *infinite games*, mathematical frameworks that can be used to model all sorts of interaction. We then recall the notion of *simulation* and its game characterisation. At the end of this chapter, we list some extended simulations that are available in the literature.

2.1 Formal Languages

A formal language, or just a language, is defined with respect to an *alphabet*. An alphabet is a set of *letters* that is usually denoted with Σ. In general, a letter can be any symbol and an alphabet can contain finitely or infinitely many letters. However in this work, we will mainly consider the usual letters instead of arbitrary symbols and only consider alphabets with finitely many letters.

A sequence of letters is called a *word* and the *length* of a word is defined as the length of the sequence. In our context, a word can have a finite or infinite length. The set of all finite words over Σ is denoted with Σ^* and the set of all infinite words over Σ is denoted with Σ^ω. The set of non-empty finite words over Σ is denoted with Σ^+, i.e. $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. We also consider the set of all finite and infinite words over Σ and denote it with Σ^∞, i.e. $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A *language* of finite or infinite words over Σ is defined as a subset of Σ^* or Σ^ω, respectively. A language of infinite words is also called an ω-language.

Example 2.1.1. Consider the alphabet $\Sigma = \{a, b\}$. The set $L = \{ab, abb, abbb, \ldots\}$ is a language of finite words over Σ and $L' = \{abbbb\ldots\}$ is an ω-language over Σ.

Throughout this work, we fix the notation regarding words and languages as follows. For any word $w \in \Sigma^\infty$, we denote by $|w|$ the length of w, and by $|w|_a$ the number of occurrences of the letter a in w. We denote by $\mathsf{Pos}(w)$ the set of *positions* in w, i.e. $\mathsf{Pos}(w) = \{1, \ldots, |w|\}$ if $w \in \Sigma^*$ and $\mathsf{Pos}(w) = \{1, 2, \ldots\}$ if $w \in \Sigma^\omega$. For any $i \in \mathsf{Pos}(w)$, we denote by $w(i)$ the letter that occurs at position i in w. For example, if $w = abbca$ then $w(2) = w(3) = b$. We denote respectively by $\mathsf{suffix}(w)$ and $\mathsf{prefix}(w)$ the sets of suffixes and prefixes of w. We denote by Σ_w the set of letters that occur in w.

One important class of languages that is widely used in many application domains is the class of *regular languages*. Given an alphabet Σ, the set of regular languages $\mathcal{L}$ is defined as the smallest set that contains the empty set $\emptyset$, the empty string language $\{\epsilon\}$, the singleton language $\{a\}$, for any $a \in \Sigma$, and for all L, L' in $\mathcal{L}$, it contains the union

$L \cup L'$, the concatenation $L \cdot L'$, and the Kleene star L^*. For example, the language L from Example 2.1.1 is regular. This is because the languages $L_1 = \{a\}$ and $L_2 = \{b\}^*$ are regular and hence by definition $L = L_1 \cdot L_2$ is also regular. We will often describe regular languages with the usual regular expression. For example, we write ab^* for the language L from Example 2.1.1.

For the infinite case we say that the ω-language $L \subseteq \Sigma^\omega$ is *ω-regular* if $L = (L_1 \setminus \{\epsilon\})^\omega$ and the language $L_1 \subseteq \Sigma^*$ is regular, or $L = L_1 \cdot L_2{}^\omega$ and the languages $L_1, L_2 \subseteq \Sigma^*$ are regular, or $L = L_1 \cup \ldots \cup L_n$ and the languages $L_1, \ldots, L_n \subseteq \Sigma^\omega$ are regular. For example, the language L' from Example 2.1.1 is regular because $\{a\}$ and $\{b\}$ are regular, and hence $L' = \{a\} \cdot \{b\}^\omega$ is ω-regular.

2.2 Decision Problems

A decision problem is a question that asks whether for some given elements $x_1, \ldots, x_n$ that can be represented with some finite data structure, a property $P(x_1, \ldots, x_n)$ holds or not [RJ87, Sip96, Koz97]. For example, given an integer $x \in \mathbb{N}^+$, check whether it is a prime number or not, or given some pairs of words $(u_1, v_1), \ldots, (u_n, v_n) \in \{0,1\}^* \times \{0,1\}^*$, check whether there are indices $i_1, \ldots, i_m$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. A decision problem can be decidable or not. By decidable this means there exists an algorithm that always terminates and give us a correct answer. For example, the first problem regarding the prime number is decidable, whereas the second one, which is known as *Post Correspondence Problem*, is not [Pos46].

2.2.1 Complexity Classes

In the theory of computation, decision problems are classified based on how difficult they are to solve. By difficult this means the computational resources that are needed by the algorithm to answer the problem. We refrain from giving the detailed explanation regarding computational complexity and its hierarchy since we will only use them to classify the complexity of our buffered simulation. For more detailed explanations, we refer to the literature [Pap94, Sip96]. The following are the classes and the notations that we will consider through out this work.

- PTIME: the class of decision problems that can be solved by a deterministic Turing machine using a polynomial amount of time.

- PSPACE: the class of decision problems that can be solved by a deterministic Turing machine using a polynomial amount of space.

- NPSPACE: the class of decision problems that can be solved by a non-deterministic Turing machine using a polynomial amount of space. Due to Savitch's theorem, this class coincides with the class PSPACE [Sav70].

- EXPTIME: the class of decision problems that can be solved by a deterministic Turing machine using an exponential amount of time, i.e. $O(2^{p(n)})$, where $p(n)$ is a polynomial function of n.

- 2EXPTIME: the class of decision problems that can be solved by a deterministic Turing machine using a doubly exponential amount of time, i.e. $O(2^{2^{p(n)}})$, where $p(n)$ is a polynomial function of n.

$$\begin{tikzcd}
 & \Pi^0_1 \arrow[dr] & & \Pi^0_2 & \cdots & {} \arrow[dr] & & \Pi^1_1 \arrow[dr] & & \Pi^1_2 & \cdots \\
\Delta^0_0 \arrow[ur] \arrow[dr] & & \Delta^0_1 \arrow[ur] \arrow[dr] & & & & \Delta^1_0 \arrow[ur] \arrow[dr] & & \Delta^1_1 \arrow[ur] \arrow[dr] & & \\
 & \Sigma^0_1 \arrow[ur] & & \Sigma^0_2 & \cdots & {} \arrow[ur] & & \Sigma^1_1 \arrow[ur] & & \Sigma^1_2 & \cdots \\
 & & \text{Arithmetical} & & & & & \text{Analytical} & & &
\end{tikzcd}$$

Figure 2.1: The arithmetical and analytical hierarchies.

Recall that the hierarchy of these classes is as follows.

$$\text{PTIME} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \subseteq \text{2EXPTIME}$$

Recall also that a problem P is called *hard* under *polynomial-time* reduction for some class C if all problems in C can be reduced in polynomial time to P. A problem P is called *complete* under *polynomial-time* reduction for C if it is in C and hard under polynomial-time reduction for C. In this thesis, since we will only consider polynomial-time reduction, we will simply call a problem P *hard* or *complete* for some class C. Intuitively, if the problem P is complete for the class C then P is considered as a maximally difficult problem in the class C.

2.2.2 Undecidability Degree

There is also a classification for undecidable problems, that is, whether one problem is more undecidable than others [RJ87, Mos80, Jap94]. In recursion theory, the classification is based on the complexity of the formula that defines the problem. The first levels of the hierarchy are called the arithmetical hierarchy, which classifies the undecidable problems using first order arithmetical formulas. It then continues with the analytical hierarchy, which classifies the undecidable problems using second order formulas. Each of these hierarchies has infinitely many classes. We illustrate them in Figure 2.1.

The classes in the arithmetical hierarchy are denoted with Σ^0_i and Π^0_i where $i \in \mathbb{N}$. For any $i \in \mathbb{N}$, we denote by Δ^0_i the intersection of Σ^0_i and Π^0_i. The lowest level in the hierarchy is Δ^0_0 which contains all decidable problems. For any $i > 0$, the classes Π^0_i and Σ^0_i are inductively defined by each time adding the universal and existential quantifiers, respectively. Hence a problem is in Σ^0_{i+1} if it can be defined as $\exists x_1 \ldots \exists x_n \,\psi$ where ψ is in Π^0_i. Similarly, a problem is in Π^0_{i+1} if it can be defined as $\forall x_1 \ldots \forall x_n \,\psi$ where ψ is in Σ^0_i. For example, if $P(x, y)$ is a decidable problem or property then $\exists x\, P(x, y)$ is in Σ^0_1 and $\forall y\, \exists x\, P(x, y)$ is in Π^0_2.

In extension to the arithmetical hierarchy, the analytical hierarchy classifies undecidable problems based on second order formulas. The classes are denoted with Σ^1_i and Π^1_i where $i \in \mathbb{N}$. For any $i \in \mathbb{N}$, we denote by Δ^1_i the intersection of Σ^1_i and Π^1_i. The lowest level in the analytical hierarchy is Δ^1_0 that contains all problems in the arithmetical hierarchy are in this class. For any $i > 0$, the classes Π^1_i and Σ^1_i are inductively defined by each time adding the universal and existential second-order quantifiers, respectively. A problem is in Σ^1_{i+1} if it can be defined as $\exists X_1 \ldots \exists X_n \psi$ where ψ is in Π^1_i. Similarly, a problem is in Π^0_{i+1} if it can be defined as $\forall X_1 \ldots \forall X_n \psi$ where ψ is in Σ^1_i.

Note that the class Π^1_1 strictly contains the whole arithmetical hierarchy, and hence if a problem is Π^1_1-complete, it is far more undecidable than the problems that are Σ^0_1 or Π^0_1-complete. Any undecidable problem that is hard for some class in the analytical hierarchy is usually called highly undecidable. In the following, we exemplify the decidability, undecidability, and high undecidability with the family of tiling problems.

2.2.3 Tiling Problems

Intuitively, we have a finite set of domino-types, i.e. squares where each side is colored with possibly different colors. For each type, we have infinitely many copies. The problem is to tile some region of a plane with dominos such that the connecting sides have the same color.

Formally, a *tiling system* $\mathcal{T} = (T, H, V)$ consists of a finite set of tiles T, a horizontal compatibility relation $H \subseteq T \times T$, and a vertical compatibility relation $V \subseteq T \times T$. A *tiling* on a region $R \subseteq \mathbb{N} \times \mathbb{N}$ is a mapping $t : R \to T$ such that $(t_{i,j}, t_{i+1,j}) \in H$ for all $(i, j), (i + 1, j) \in R$ and $(t_{i,j}, t_{i,j+1}) \in V$ for all $(i, j), (i, j + 1) \in R$.

Example 2.2.1. Let $\mathcal{T} = (T, H, V)$ be a tiling system where $T = \{t_1, t_2, t_3\}$, $H = V = \{(t_1, t_2), (t_2, t_1), (t_2, t_3), (t_3, t_2)\}$ and $R = \mathbb{N}^+ \times \mathbb{N}^+$. Let $t : R \to T$, where

$$t(x, y) = \begin{cases} t_1, & \text{if } x, y \text{ even} \\ t_3, & \text{if } x, y \text{ odd} \\ t_2, & \text{otherwise.} \end{cases}$$

The function t is a tiling on R with respect to $\mathcal{T}$. The first row is tiled with $t_3t_2t_3t_2\ldots$, the second row with $t_2t_1t_2t_1\ldots$, the third row with $t_3t_2t_3t_2\ldots$, and so on.

The problem of deciding whether for a given tiling system $\mathcal{T}$, there exists a tiling for the first quadrant of Cartesian plane, i.e. $\mathbb{N}^+ \times \mathbb{N}^+$, is known as the *unbounded tiling problem*. This problem was first introduced in [Wan60]. It is known to be undecidable and complete for the class Π^0_1.

Proposition 2.2.2 ([Wan60])**.** *The unbounded tiling problem is* Π^0_1*-complete.*

Equivalently, the problem of deciding whether for a given tiling system $\mathcal{T}$, there exists a tiling for the octant of Cartesian plane, i.e. $\{(i, j) \in \mathbb{N}^+ \times \mathbb{N}^+ \mid i \leq j\}$, is also undecidable and complete for the class Π^0_1. Such a problem is also known as the *octant tiling problem*.

Proposition 2.2.3 ([BGG97])**.** *The octant tiling problem is* Π^0_1*-complete.*

There is a decidable variant of tiling problem called *corridor tiling problem*. In this case, we are given a tiling system $\mathcal{T} = (T, H, V)$ and a size of corridor $n > 0$ encoded unarily. We are then asked whether there exists a tiling t on the region $\{1, \ldots, n\} \times \mathbb{N}^+$. The problem of solving the corridor tiling problem is known to be PSPACE-complete [Boa97, Chl86].

There is also a variant of corridor tiling where besides the tiling system $\mathcal{T} = (T, H, V)$ and a size of corridor $n > 0$ encoded unarily, we are also given some initial and final tiles

$t_I, t_F \in T$. We are then asked whether there exists $m > 0$ such that we have a tiling on $\{1, \ldots, n\} \times \{1, \ldots, m\}$ that uses the initial tile t_I as the first tile and the final tile t_F as the last tile, i.e. $t(1,1) = t_I$ and $t(n,m) = t_F$. Such a variant of corridor tiling is also known to be PSPACE-complete.

Proposition 2.2.4 ([Boa97, Chl86]). *Both variants of the corridor tiling problem are PSPACE-complete.*

Furthermore, there is a variant of tiling problem that is highly undecidable called *recurrent tiling problem*. This problem extends the unbounded and octant tiling problems by additionally asking for infinite occurrences of a certain tile in the first column. Given a tiling system $\mathcal{T} = (T, H, V)$ with a designated final tile $t_F \in T$, the problem asks whether there exists a tiling t on $\mathbb{N}^+ \times \mathbb{N}^+$, or equivalently in $\{(i, j) \in \mathbb{N}^+ \times \mathbb{N}^+ \mid i \leq j\}$, such that there exist infinitely many j with $t(1, j) = t_F$.

Proposition 2.2.5 ([Har85]). *Both the recurrent unbounded and recurrent octant tiling problems are Σ^1_1-complete.*

We will use this family of tiling problems later and use them to show the hardness of buffered simulation.

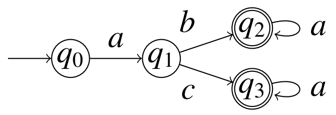
2.3 Automata

An automaton is a mathematical model that is often used to reason about computations of a system or program. Intuitively, it is an abstract machine that runs on a given input word by going through a sequence of states and either accepts or rejects the input word. An automaton visually can be seen as a directed graph where each edge is labeled and some nodes are considered to be final or accepting. The node represents the state of the system and an edge that is labeled with a represents the state-changing of the system when the action a is executed. Based on the length of the input word, automata are classified into two kinds: *finite* or *ω-automata*.

2.3.1 Finite Automata

Formally, a finite automaton is defined as a tuple $\mathcal{A} = (Q, \Sigma, E, q_I, F)$ where Q is a set of states, Σ is an alphabet, $q_I \in Q$ is a starting state, $E \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is the set of final states. We usually write $p \xrightarrow{a} p'$ if $(p, a, p) \in E$. For any finite word $w = a_1 \ldots a_n \in \Sigma^*$, a *run* r of an automaton $\mathcal{A}$ is an alternating sequence of states and letters, i.e. $r = q_0 a_1 q_1 \ldots a_n q_n$, such that for all $i \in \{1, \ldots, n\}$, $q_{i-1} \xrightarrow{a_i} q_i$. The run r is called *accepting* if $q_0 = q_I$ and $q_n \in F$, and such a word $w = a_1 \ldots a_n$ is said to be *accepted* by the automaton $\mathcal{A}$. The set of words that are accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$ and it is called the language of $\mathcal{A}$. The size or cardinality of $\mathcal{A}$ is the number of states in $\mathcal{A}$ and it is denoted by $|\mathcal{A}|$. The automaton $\mathcal{A}$ is called *deterministic* if for any state $q \in Q$ and letter $a \in \Sigma$, there is a unique state $q' \in Q$ such that $q \xrightarrow{a} q'$. Moreover, for any run $r = p_0 a_1 p_1 \ldots a_n p_n$, we denote by $\mathsf{word}(r)$ the word $a_1 \ldots a_n$.

Example 2.3.1. Consider the following automaton $\mathcal{A}$ over the alphabet $\Sigma = \{a, b, c\}$.



The automaton $\mathcal{A}$ is non-deterministic since from q_0 there is no $q \in Q^{\mathcal{A}}$ such that $q_0 \xrightarrow{b} q'$. Moreover, $r = q_0aq_1bq_2$ and $r' = q_0aq_1cq_3aq_3$ are some examples of accepting runs in $\mathcal{A}$. The language of $\mathcal{A}$ is $L(\mathcal{A}) = aba^* \cup aca^*$.

The class of the languages that are accepted by finite automata coincides with the class of regular languages.

Proposition 2.3.2 ([Kle56]). *Let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is regular iff there is a finite automaton $\mathcal{A}$ such that $L = L(\mathcal{A})$.*

This allows many decision problems regarding regular languages such as inclusion or universality to be solved using algorithms on graphs. For example, one standard way to check universality of a language L recognised by $\mathcal{A}$ is by checking whether its complement, i.e. the automaton $\overline{\mathcal{A}}$ that recognises $\Sigma^* \setminus L(\mathcal{A})$, accepts the empty language. Unfortunately, such an automaton $\overline{\mathcal{A}}$ is exponentially larger than $\mathcal{A}$.

Complementation of NFA

For any NFA $\mathcal{A}$, there is a deterministic finite automaton (DFA) $\overline{\mathcal{A}}$ that recognises the complement of L with size at most $2^{|\mathcal{A}|}$. We can constuct such a DFA with the subset construction which basically converts $\mathcal{A}$ to a DFA that recognises the same language L and then invert its final states [RS59, HMU06].

Proposition 2.3.3 ([RS59]). *For any NFA $\mathcal{A}$ over Σ, there is an NFA $\overline{\mathcal{A}}$ of size $O(2^n)$ such that $L(\overline{\mathcal{A}}) = \Sigma^* \setminus L(\mathcal{A})$.*

This upper bound is known to be optimal. We cannot complement an NFA with a blow up better than $O(2^n)$ [SS78, Bir93].

Inclusion of NFA

For any two NFA, deciding language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is computationally hard. It is PSPACE-complete [MS73]. It is in PSPACE since to decide inclusion, we have to check whether the intersection of the language of $\mathcal{A}$ and the complement of the language of $\mathcal{B}$ is empty, i.e. whether $L(\mathcal{A}) \cap \overline{L(\mathcal{B})} = \emptyset$. It is possible to construct an NFA that recognises $L(\mathcal{A}) \cap \overline{L(\mathcal{B})}$ and by Proposition 2.3.3 such an NFA is exponentially larger than $\mathcal{B}$. We can then test it for emptiness. Checking the emptiness for finite automata is NLOGSPACE-complete: we can use the depth-first-search algorithm to check whether there is a final state that is reachable from the initial state. This then yields an NPSPACE procedure and by Savitch theorem [Sav70], the problem is also in PSPACE.

This upper bound is optimal. Checking language inclusion between two NFA is also PSPACE-hard. For example, we can polynomially reduce the corridor tiling problem to the problem of deciding language non-inclusion $L(\mathcal{A}) \nsubseteq L(\mathcal{B})$. Given a tiling system $\mathcal{T} = (T, H, V)$, initial and final tiles t_I, t_F, and a width n, we construct two NFA $\mathcal{A}$, $\mathcal{B}$ over T such that the automaton $\mathcal{A}$ accepts any word $w \in (T^n)^+$ where the first and the last letters are t_I and t_F, and $\mathcal{B}$ accepts such a word w if there is a vertical or horizontal mismatch. We have language non-inclusion $L(\mathcal{A}) \nsubseteq L(\mathcal{B})$ iff there exists a valid corridor tiling.

Proposition 2.3.4 ([MS73])**.** *For any two NFA $\mathcal{A}$, $\mathcal{B}$, deciding $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is PSPACE-complete.*

Finite automata are suitable to model systems or programs that always terminate such as compilers, parsers, search engines, etc. [ASU86, HMU06]. Such a system is designed to run only for some period of time and then gives an output. For systems that are designed to run forever such as operating systems, communication protocols, or web servers, one usually considers *ω-automata* [VW94, Tho90], an extended notion of finite automata that accept ω-languages.

2.3.2 ω-Automata

We will start with the simplest form of ω-automata called *(non-deterministic) Büchi automata*. Non-deterministic Büchi automata, or shortly NBA, were introduced in the 1960's as an auxiliary device to obtain a decision procedure for monadic-second-order logic [Büc62]. However, they are now used in many applications such as in system verification to model reactive systems and specify their non-terminating behaviour [EH00, Hol04] and in termination analysis of recursive programs [LJBA01, FV09]. The definition of NBA basically does not differ from the definition of finite automata. It is also a quintuple $\mathcal{A} = (Q, \Sigma, E, q_I, F)$. The difference is that a run is an infinite alternating sequence $\rho = q_0 a_1 q_1 a_2 \ldots$ where for all $i \in \mathbb{N}$, $(q_i, a_{i+1}, q_{i+1}) \in E$. Such a run ρ is accepting if $q_0 = q_I$ and there exist infinitely many i such that $q_i \in F$. Hence we can see NBA as the acceptors of ω-languages. For example, consider again the automaton $\mathcal{A}$ from Example 2.3.1. If we see it as a Büchi automaton instead of a finite automaton, the language of $\mathcal{A}$ is $L(\mathcal{A}) = aba^\omega \cup aca^\omega$.

Let us denote with $\mathsf{Run}(\mathcal{A})$ the set of infinite runs of $\mathcal{A}$ and $\mathsf{AccRun}(\mathcal{A})$ the set of accepting runs of $\mathcal{A}$.

The class of languages that are accepted by NBA coincides with the class of ω-regular languages.

Proposition 2.3.5 ([Büc62])**.** *Let Σ be an alphabet. For any language $L \subseteq \Sigma^*$, L is ω-regular iff there is a Büchi automaton $\mathcal{A}$ over Σ such that $L = L(\mathcal{A})$.*

Many decision problems regarding non-terminating reactive systems can be solved using an automata-theoretic approach. For example, the problem of checking whether the behaviour of a reactive system complies to some specification is typically modeled as the inclusion problem between two NBA in which one represents the system and the other one represents the specification [VW86, Var96]. Unfortunately, the same as in the case of finite automata, solving language inclusion between two NBA is expensive. We need to perform complementation for one of the input automata.

Complementation of NBA

Given an NBA $\mathcal{A}$ over Σ, we can construct an NBA $\overline{\mathcal{A}}$ that recognises the complement of $L(\mathcal{A})$. In general, the automaton $\overline{\mathcal{A}}$ is also exponentially larger than $\mathcal{A}$ [Büc62, SVW87]. We will show the complementation procedure for NBA that uses Ramsey's theorem from the combinatorics area [Ram30]. The reason is because in Chapter 4, we will use a similar approach to show decidability of a special case of buffered simulation.

First let us denote with $q_0 \xrightarrow{a_1 \ldots a_n} q_n$ if there exists a finite run $q_0 a_1 q_1 \ldots q_n$ in $\mathcal{A}$ and write $q_0 \xrightarrow[F]{a_1 \ldots a_n} q_n$ if such a run goes through an accepting state, i.e. there exists

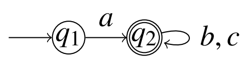
$i \in \{1, \ldots, n\}$ such that $q_i \in F$. We say that two finite words u, $v \in \Sigma^*$ are equivalent if they behave the same in the automaton $\mathcal{A}$.

Definition 2.3.6 ([Büc62])**.** Given an NBA $\mathcal{A} = (Q, \Sigma, E, q_I, F)$, two words $u, v \in \Sigma^*$ are equivalent and we write $u \approx v$, if for every $p, q \in Q$,

- $p \xrightarrow{u} q$ iff $p \xrightarrow{v} q$, and
- $p \xrightarrow[F]{u} q$ iff $p \xrightarrow[F]{v} q$.

For any word $u \in \Sigma^*$, we denote by $[u]$ the equivalence class of u, i.e. $[u] = \{w \in \Sigma^* \mid w \approx u\}$, and with $\Sigma^*/\approx$, the set of all equivalence classes induced by $\approx$.

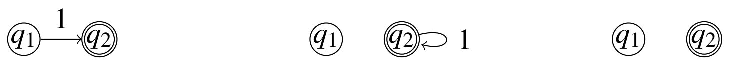
Example 2.3.7. Consider the following NBA $\mathcal{A}$ over $\Sigma = \{a, b, c\}$.



We have $ab \approx ac$ since by reading ab or ac, we can go from state q_1 to q_2. We also see an accepting state whenever we form such paths. In fact, every finite word that is started with a and followed by finitely many b or c is equivalent to ab and ac. We have $[ab] = [ac] = a(b \cup c)^*$.

For any NBA $\mathcal{A}$ and a word $u \in \Sigma^*$, let us define the *transition profile* of u as a labeled graph $G = (V, E, f)$ where the nodes are $V = Q^{\mathcal{A}}$, the edges are $E = \{(p, q) \in Q^{\mathcal{A}} \times Q^{\mathcal{A}} \mid p \xrightarrow{u} q\}$, and for any edge (p, q), the labeling is $f(p, q) = 1$ if $p \xrightarrow[F]{u} q$ and $f(p, q) = 0$ otherwise.

Example 2.3.8. Consider the NBA $\mathcal{A}$ over $\Sigma = \{a, b, c\}$ from Example 2.3.7. Let $u_1 = ab$, $u_2 = bc$, and $u_3 = aa$. The following are the transition profiles of u_1, u_2, and u_3, respectively.



Intuitively, the edge in the transition profile of u represents the path that can be formed by reading u in $\mathcal{A}$ and the label tells us whether the path sees a final state or not. Hence for any two equivalent words $u \approx u'$, their transition profiles are the same. Any equivalence class in $\Sigma^*/\approx$ is indeed uniquely determined by a transition profile. We can also say that G_u is a transition profile of the class $[u]$ instead of the word u. Moreover, since there are at most $3^{|\mathcal{A}|^2}$ different transition profiles, we have the following proposition.

Proposition 2.3.9. *For any NBA $\mathcal{A}$, $|\Sigma^*/\approx| \leq 3^{|\mathcal{A}|^2}$.*

Let $\mathcal{A}$ be some NBA and $[u]$ in $\Sigma^*/\approx$. We can construct a DFA $\mathcal{A}'$ that recognises $[u]$ [SVW87]. The technique is similar to the subset construction, but in an extended way. Suppose $Q^{\mathcal{A}} = \{q_1, \ldots, q_n\}$ are the set of states in $\mathcal{A}$. A state in the DFA $\mathcal{A}'$ then is of the form $(P_1, \ldots, P_n)$ where $P_i \subseteq Q^{\mathcal{A}} \times \{0, 1\}$ for all $i \in \{1, \ldots, n\}$. We remember n-tuples of sets instead of a single set because we need to capture information about runs that can start from any state in $\mathcal{A}$. Each component $P_1, \ldots, P_n$ in the state of $\mathcal{A}'$ simulates runs that are started from $q_1, \ldots, q_n$ respectively. We then remember whether each run has seen an accepting state by using 0 or 1. A run in $\mathcal{A}'$ then is accepting if it represents runs that start and end in the corresponding state according to the transition profile of $[u]$.

Proposition 2.3.10 ([SVW87]). *Let $\mathcal{A}$ be an NBA with n states. For any $[u] \in \Sigma/\approx$, there is a DFA $\mathcal{A}'$ where $|\mathcal{A}'| \leq 4^{n^2}$ such that $L(\mathcal{A}') = [u]$.*

For any infinite word $w \in \Sigma^\omega$, there are two equivalence classes $[u], [v] \in \Sigma^*/\approx$ such that $w \in [u] \cdot [v]^\omega$ [Büc62]. We can even find such a *proper* pair of equivalence classes, that is, the one that satisfies $[uv] = [u]$ and $[vv] = [v]$ [FV10]. We can show this by using the well-known Ramsey's theorem from the combinatorics area. For notational purpose, let us denote with $[[S]]^2$, the set of all subsets of S with cardinality 2, i.e.

$$[[S]]^2 = \{\, \{i, j\} \,|\, i, j \in S,\ i \neq j\,\}.$$

We can state Ramsey's theorem as follows.

Proposition 2.3.11 ([Ram30, Ros81]). *Let $\langle P_1, \ldots, P_n\rangle$ be a partition of $[[\mathbb{N}]]^2$. There is an infinite set $X \subseteq \mathbb{N}$ and $i \in \{1, \ldots, n\}$ such that $[X]^2 \subseteq P_i$.*

Intuitively, this proposition says that if we have a complete graph, in which the nodes are natural numbers, and we color every edge with some number between 1 to n, then there must be a subgraph over infinitely many nodes where all of its edges have the same color. We can use this proposition to show the following characterisation of infinite words.

Proposition 2.3.12 ([Büc62]). *For every NBA $\mathcal{A}$ over Σ and $w \in \Sigma^\omega$ there is a proper pair of equivalence classes $[u]$, $[v] \in \Sigma/\approx$ such that $w \in [u] \cdot [v]^\omega$.*

Proof. Let $w = a_1 a_2 \ldots$ Together with the relation $\approx$, the word w defines a partition of $[[\mathbb{N}]]^2$. Let $\Sigma^*/\approx\, = \{[u_1], \ldots, [u_n]\}$ and

$$P_k = \{\{i, j\} \mid i < j \text{ and } a_i \ldots a_{j-1} \in [u_k]\},$$

for $k \in \{1, \ldots, n\}$. Hence P_k contains any pair of i, j where $a_i \ldots a_{j-1} \in [u_k]$. Since any word $a_i \ldots a_{j-1}$ belongs to one of $[u_1], \ldots, [u_n]$ and $[u_1], \ldots, [u_n]$ are disjoint, $\langle P_1, \ldots, P_n\rangle$ is a partition of $[[\mathbb{N}]]^2$.

By Proposition 2.3.11, there is an infinite set $X \subseteq \mathbb{N}$ and an equivalence class $[v] \in \Sigma^*/\sim$ such that for all $i, j \in X$, we have $a_i \ldots a_{j-1} \in [v]$. Let $i_1, i_2, \ldots$ be the infinite increasing sequence in X. Hence the word w is of the form $w_0 w_1 w_2 \ldots$ where $w_k = a_{i_k} a_{i_k+1} \ldots a_{i_{k+1}}$ for $k > 0$. Let $[u]$ be the equivalence class of $w_0 w_1$. Since $w_2, w_3, \ldots \in [v]$, we have $w \in [u] \cdot [v]^\omega$.

To show that the pair $[u]$, $[v]$ is proper, first note that $u \approx w_0 w_1$ and $v \approx w_2$. Hence $uv \approx w_0 w_1 w_2$. Since $w_1 \approx w_1 w_2$ and $w_0 w_1 \approx u$, we have $uv \approx u$. Thus $[uv] = [u]$. Moreover, note also that $w_1 \approx v$, $w_2 \approx v$, and $w_1 w_2 \approx v$. This nonetheless implies that $vv \approx v$. Thus $[vv] = [v]$. □

Moreover, if we consider a pair of equivalence classes $[u], [v] \in \Sigma^*/\approx$, either the ω-language $[u]\cdot[v]^\omega$ is included in $L(\mathcal{A})$ or included in the complement of $L(\mathcal{A})$. To illustrate this, suppose we have a word $w \in [u] \cdot [v]^\omega$. Hence $w = w_0 w_1 \ldots$ where $w_0 \in [u]$ and $w_1, w_2, \ldots \in [v]$. If $w \in L(\mathcal{A})$ then there are $p_0, p_1, \ldots$ such that

$$p_0 \xrightarrow{w_0} p_1 \xrightarrow{w_1} p_2 \ldots$$

and infinitely many i such that $p_i \xrightarrow[F]{w_i} p_{i+1}$. Now let $w' \in [u] \cdot [v]^\omega$ be an arbitrary word. We have $w' = w'_0 w'_1 \ldots$ where $w'_0 \in [u]$ and $w'_1, w'_2, \ldots \in [v]$. Moreover, since $w_i \approx w'_i$ for all $i \geq 0$, we have

$$p_0 \xrightarrow{w'_0} p_1 \xrightarrow{w'_1} p_2 \ldots$$

and there exist infinitely many i such that $p_i \xrightarrow[F]{} p_{i+1}$. Thus w' is also accepted by $\mathcal{A}$. Since w' is arbitrary, this shows $[u] \cdot [v]^\omega \subseteq L(\mathcal{A})$. In the case of $w \notin L(\mathcal{A})$, we have $[u] \cdot [v]^\omega \subseteq \overline{L(\mathcal{A})}$. Since otherwise, there is a word $w'' \in [u] \cdot [v]^\omega$, but $w'' \in L(\mathcal{A})$. We have shown that this would imply $w \in L(\mathcal{A})$, and contradicts our initial assumption.

Proposition 2.3.13 ([Büc62]). *For any $[u], [v] \in \Sigma^*/\approx$,*

- *if $[u] \cdot [v]^\omega \cap L(\mathcal{A}) \neq \emptyset$ then $[u] \cdot [v]^\omega \subseteq L(\mathcal{A})$.*
- *if $[u] \cdot [v]^\omega \cap \overline{L(\mathcal{A})} \neq \emptyset$ then $[u] \cdot [v]^\omega \subseteq \overline{L(\mathcal{A})}$.*

We can construct an NBA that recognises the complement of the language of $\mathcal{A}$ as follows. Suppose $\Sigma^*/\approx \; = \{[u_1], \ldots, [u_n]\}$. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be the corresponding finite automata that recognise $[u_1], \ldots, [u_n]$ respectively. By Proposition 2.3.10, each of the automata at most has $4^{|\mathcal{A}|^2}$ states. Now for each $[u_i], [u_j] \in \Sigma^*/\approx$ in which the intersection of $[u_i] \cdot [u_j]^\omega$ with $L(\mathcal{A})$ is empty, we construct an NBA $\mathcal{A}_{ij}$ that recognises $[u_i] \cdot [u_j]^\omega$ from $\mathcal{A}_i$ and $\mathcal{A}_j$. We then construct the NBA $\overline{\mathcal{A}}$ that recognises the complement of $L(\mathcal{A})$ by taking the union of all such NBA.

Proposition 2.3.14 ([SVW87]). *For any NBA $\mathcal{A}$, there is an NBA $\overline{\mathcal{A}}$ of size $O(16^{|\mathcal{A}|^2})$ such that $L(\overline{\mathcal{A}}) = \overline{L(\mathcal{A})}$.*

This upper bound, however, is not the optimal one. In the literature, there is a more involved procedure that constructs the complement of an NBA $\mathcal{A}$ with the blow-up of $O((0.76 \cdot |\mathcal{A}|)^{|\mathcal{A}|})$ [Sch09] which is known to be the optimal one [Yan08]. Nevertheless, we cannot avoid the exponential blow-up when complementing an NBA [Saf88, HL11].

Language Inclusion of NBA

As in the case of finite automata, checking language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ between two NBA $\mathcal{A}, \mathcal{B}$ is also PSPACE-complete. We have to check whether the intersection of the language of $\mathcal{A}$ and the complement of the language of $\mathcal{B}$ is empty. The NBA that recognises $L(\mathcal{A}) \cap \overline{L(\mathcal{B})}$ is exponentially larger than $\mathcal{B}$. However, checking emptiness for an NBA is NLOGSPACE-complete. We can check with a depth-first-search whether there is a final state p that is reachable from the initial state and reachable from itself. This yields an NPSPACE procedure and by Savitch's theorem [Sav70], solving such a problem is also in PSPACE.

Checking language inclusion between two NBA is also PSPACE-hard. We can lift the hardness of solving language inclusion problem between the NFA.

Proposition 2.3.15 ([Büc62, SVW87]). *For any two NBA $\mathcal{A}, \mathcal{B}$, deciding $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is PSPACE-complete.*

Other ω-Regular Automata

Besides Büchi automata, there are also other ω-automata that coincide with the ω-regular languages such as *generalised Büchi automata* (GNBA) and *parity automata* [Tho90]. Their definitions do not differ significantly from Büchi automata. The only difference is in the last component that defines the acceptance of a run. Before we give their definition, first for any infinite sequence $a_1a_2\ldots$ where $a_1, a_2, \ldots \in S$, let us denote with $\mathsf{inf}(a_1a_2\ldots)$ the set of elements of S that infinitely occur in $a_1a_2\ldots$.

- In a GNBA, instead of a set of final states $F \subseteq Q$, the acceptance is defined with respect to a set of sets of states $\{F_1, \ldots, F_n\} \subseteq 2^Q$. We say that a run $r = q_0 a_1 q_1 a_2 \ldots$ is accepting iff for each $i \in \{1, \ldots, n\}$, there is a state $q \in F_i$ that is seen infinitely often in r, i.e. $\mathsf{inf}(q_0 q_1 \ldots) \cap F_i \neq \emptyset$.

- In a parity automaton, the acceptance is defined with respect to a priority function $\Omega : Q \to \mathbb{N}$ that assigns a priority to each state. We say that a run $r = q_0 a_1 q_1 a_2 \ldots$ is accepting iff the highest priority that is seen infinitely often in r is even, i.e. $\mathsf{max}(\mathsf{inf}(\Omega(q_0)\Omega(q_1)\ldots))$ is even.

These are ω-automata other than Büchi automata that we will consider in this work. Note that Büchi automata can be seen as a special case of GNBA and parity automata. A Büchi automaton $\mathcal{A}$ with the set of final states F can be seen as a parity automaton with a priority function $\Omega : Q \to \{1, 2\}$ where $\Omega(q) = 2$ if $q \in F$ and 1 if $q \notin F$, and as a GNBA with acceptance $\{F\}$.

We can also translate both GNBA and parity automata to Büchi automata. For any GNBA $\mathcal{A}$ with acceptance $\{F_1, \ldots, F_n\}$, we can construct an NBA $\mathcal{A}'$ that is obtained from $\mathcal{A}$ by additionally having a counter $c \in \{1, \ldots, n, \top\}$ in its state space. The counter is used to remember whether we have seen at least one state from $F_1, \ldots, F_n$. Once we have seen all of them then we reset the counter. In the automaton $\mathcal{A}'$, the states where we reset the counter are considered to be accepting. Hence an accepting run in the NBA $\mathcal{A}'$ simulates the one in GNBA $\mathcal{A}$.

Proposition 2.3.16 ([Tho90])**.** *For any GNBA $\mathcal{A}$ with n states, there is an NBA $\mathcal{A}'$ of size $O(n^2)$ such that $L(\mathcal{A}) = L(\mathcal{A}')$.*

For a parity automaton $\mathcal{A}$ with priority $0, 1, \ldots, n$, we can construct an NBA $\mathcal{A}'$ that guesses the maximal priority that will be seen infinitely often. Intuitively, the automaton $\mathcal{A}'$ consists of the original states from $\mathcal{A}$ and $\lfloor \frac{n}{2} \rfloor + 1$ copies of $\mathcal{A}$, namely $\mathcal{A}_0$, …, $\mathcal{A}_{\lfloor \frac{n}{2} \rfloor}$, that are reachable from the original states of $\mathcal{A}$. Each $\mathcal{A}_i$ contains states with priority at most i and the states in $\mathcal{A}_i$ with priority i are considered to be accepting. Hence to simulate an accepting run ρ of $\mathcal{A}$ in which i is the maximal priority that is seen infinitely often, we first go through the states in the original copy of $\mathcal{A}$. Since after some k steps, we will never visit a state with priority more than i, then after k steps, we move to $\mathcal{A}_i$ and simulate ρ from there. Such a run is accepting in $\mathcal{A}'$. We can simulate any accepting run in the parity automaton $\mathcal{A}$ in the NBA $\mathcal{A}'$.

Proposition 2.3.17 ([Tho90])**.** *For any parity automaton $\mathcal{A}$ with n states and m priorities, there is an NBA $\mathcal{A}'$ of size $O(nm)$ such that $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proposition 2.3.16 and Proposition 2.3.17 show that we can polynomially translate GNBA and parity automata to Büchi automata. Hence the inclusion problem between GNBA or parity automata can be reduced to the one between Büchi automata. Deciding language inclusion between such automata is in PSPACE. It is also PSPACE-hard since Büchi automata are a special case of GNBA and parity automata.

Proposition 2.3.18 ([MS73])**.** *For any two GNBA or parity automata $\mathcal{A}, \mathcal{B}$, the problem of deciding whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is PSPACE-complete.*

Through out this work, we will mostly consider Büchi automata. However in some part of this work, we will consider language inclusion between GNBA and parity automata.

2.4 Infinite Games

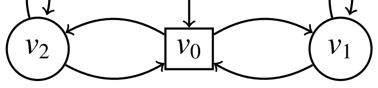
An *infinite two-player game* is a mathematical framework that models the ongoing interaction between two agents that possibly lasts in infinite duration. An infinite two-player game or just *game* is basically composed of two components: a *configuration graph* and a *winning condition*. A configuration graph is the arena where the game takes place and a winning condition is the component that determines the winner of the game.

In its formal definition, a game is a tuple $\mathcal{G} = (G, v_0, \mathsf{Win})$ where $G = (V, V_0, V_1, E)$ is a configuration graph, (V, E) is a finite or infinite directed graph, $\langle V_0, V_1 \rangle$ is a partition of V into nodes that belong to player 0 and player 1, v_0 is the starting node, and Win is the winning condition. The game is played by two players, usually called player 0 (female) and player 1 (male). Intuitively, a play proceeds as follows. Initially, a pebble is placed in the starting node. In each round, if the pebble is in the node of player $i \in \{0, 1\}$ then player i moves the pebble one step to some successor node. If he or she cannot do so then the opponent wins. Otherwise we proceed to the next round. Hence either eventually one of the players gets stuck, and in this case the opponent wins, or the play goes on for infinitely many rounds and both of the players form an infinite path in the configuration graph. We formally call such a path a *play*. The set of plays in $\mathcal{G}$ is denoted by $\mathsf{Play}(\mathcal{G})$, or simply Play, and it consists of all infinite words $\pi = v_0 v_1 \ldots \in V^\omega$ such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. We determine whether player 0 wins the play π by looking at the winning condition Win. In general, the winning condition Win is a subset of Play. Player 0 wins an infinite play π iff π is in Win.

We can also consider some well-known ω-regular conditions to define the winning condition of a game. For example, we can consider Büchi or parity conditions that are explicitly defined in the previous section. With a Büchi condition, Win is a set $F \subseteq V$, and player 0 wins the play $\pi = v_0 v_1 \ldots$ iff there exist infinitely many i such that $v_i \in F$. With parity condition, Win is a priority function $\Omega : V \to \mathbb{N}$ that assigns a priority to each node and player 0 wins the play $\pi = v_0 v_1 \ldots$ iff the maximum priority that is seen infinitely often in π is even.

We say that a player *wins* the game $\mathcal{G}$ if no matter how the opponent plays, he or she can win the play in $\mathcal{G}$. One often uses the notion of *strategy* in order to formally define this. A strategy of player $i \in \{0, 1\}$ is a partial function $\sigma_i : V^* \cdot V_i \to V$ such that for every finite play $r = v_0 \ldots v_n$, we have $(v_n, \sigma(r)) \in E$. Intuitively, a strategy σ_i tells player i where to move the pebble. We say a play $v_0 v_1 \ldots$ is *played according* to σ_i if for every $j \in \mathbb{N}$, whenever $v_j \in V_i$, we have $\sigma_i(v_0 \ldots v_j) = v_{j+1}$. A strategy σ_i of player i is called *winning* if player i wins any play π that is played according to σ_i. If a player has a winning strategy then he or she wins the game $\mathcal{G}$.

Example 2.4.1. Consider a parity game $\mathcal{G} = ((V, V_0, V_1, E), v_0, \Omega)$ where the game graph is depicted as follows.



We denote the nodes that belong to player 0 by circle and the ones that belong to player 1 by square. Consider a priority function $\Omega : V \to \{0, 1, 2\}$ where $\Omega(v_i) = i$. Player 1 wins the game $\mathcal{G}$ since he has a winning strategy σ_1 where for every finite play r, $\sigma_1(r v_0) = v_1$. In other words, whenever the pebble arrives in v_0, player 1 moves it to v_1. No matter how player 0 plays, we always obtain a play in which the highest priority that is seen infinitely often is 1. Thus player 1 wins the game $\mathcal{G}$.

The winning strategy for Player 1 that we have given in Example 2.4.1 is of a special kind. Player 1 basically decides where to move the pebble based only on the current position of the pebble and not from the history of the play. Such a strategy is called *memoryless*. Formally, a strategy σ_i for player $i \in \{0, 1\}$ is called memoryless if for any $v \in V_i$ and two finite plays that end in v, i.e. $r = v_0 v_1 \dots v$ and $r' = v'_0 v'_1 \dots v$, we have $\sigma_i(r) = \sigma_i(r')$.

There is a nice result regarding the determinacy of parity games. Parity games are *memoryless-determined* [EJ91]. Determined means that either player 0 or player 1 wins the game, and memoryless means that the player that wins has a memoryless winning strategy. Furthermore, in the case where the parity game is played on a finite graph, the problem of deciding the winner of a parity game can be solved in time polynomially in its size and exponentially in the number of priorities [Jur00].

In this work, we will mostly consider parity games with three priorities 0, 1, and 2. Such a parity game $\mathcal{G}$ with $|V|$ many nodes and $|E|$ many edges can be solved in time $\mathcal{O}(|V| \cdot |E|)$.

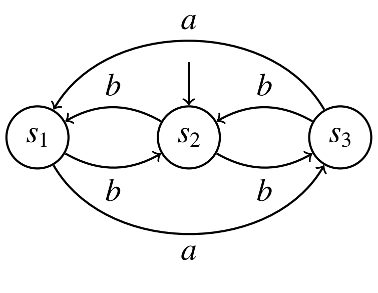
Proposition 2.4.2 ([Jur00])**.** *Let $\mathcal{G} = ((V, V_0, V_1, E), v_0, \Omega)$ be a parity game with priorities 0, 1, and 2. The problem of deciding whether player 0 wins $\mathcal{G}$ can be done in time $\mathcal{O}(|V| \cdot |E|)$.*

Throughout this work, we will frequently refer to this proposition. We will often reduce some decidability questions of buffered simulation to the problem of deciding the winner of a parity game with priorities 0, 1, and 2.

2.5 Simulation

Originally, simulation was introduced as a pre-order relation between states in a *labeled transition system* [Mil71]. Recall that a labeled transition system (LTS) is a labeled directed graph $L = (S, I, Act, \rightarrow)$ where S is a set of states, $I \subseteq S$ is a set of initial states, Act is a set of actions, and $\rightarrow \subseteq S \times Act \times S$ is an edge relation. We write $s \xrightarrow{a} t$ if $(s, a, s') \in \rightarrow$. Intuitively, simulation relates two states s, s' in an LTS if s' can mimic all stepwise behaviour of s. For an LTS $L = (S, I, Act, \rightarrow)$, let $R \subseteq S \times S$ be a binary relation such that for any pair $(s_1, s_2) \in R$, action $a \in Act$, and state $s_1{}' \in S$, if $s_1 \xrightarrow{a} s_1{}'$ then there is $s_2{}' \in S$ such that $s_2 \xrightarrow{a} s_2{}'$ and $(s'_1, s'_2) \in R$. Simulation then is defined as the greatest such relation R.

Example 2.5.1. Consider the following LTS $L = (S, I, Act, \rightarrow)$ where $S = \{s_1, s_2, s_3\}$, $I = \{s_2\}$, $Act = \{a, b\}$, and the relation $\rightarrow$ is illustrated as follows.



In this case, the simulation relation is $R = \{(s_1, s_1), (s_1, s_3), (s_2, s_2), (s_3, s_1), (s_3, s_3)\}$.

There is a simple game characterisation to check whether a state s' simulates a state s. The game is played between two players called Spoiler (male) and Duplicator (female). Initially, two pebbles, owned by Spoiler and Duplicator, are placed each in s and s', respectively. In every round, Spoiler moves his pebble one step by reading an action and Duplicator tries to mimic this by moving her pebble one step and reading the same action. This then continues round by round. If one of the players eventually gets stuck then we declare the opponent to be the winner. Otherwise, the play goes on infinitely many rounds, and in this case Duplicator wins. It is not hard to see that the state s' simulates s, i.e. $(s, s') \in R$, iff Duplicator wins the simulation game.

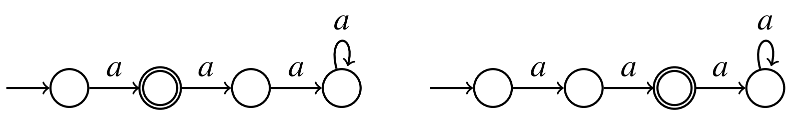
The notion of simulation can be adapted to relate the states of an automaton. We can extend the notion of simulation and use it for quotienting automata [BG03, GBS02, EWS01], approximating language inclusion [DHWT92, FW05, ABH+08], pruning transitions [ACC+11, ACC+10], and improving existing decision procedures on NBA like the Ramsey-based one [FV09] or the antichain algorithm for inclusion and universality checking [DR09].

In this work, we will consider simulation for approximating language inclusion between two NBA $\mathcal{A}$ and $\mathcal{B}$. We check whether the initial state of $\mathcal{B}$ simulates the one of $\mathcal{A}$. We will consider the simulation game where Spoiler's and Duplicator's pebbles are initially placed in the initial state of $\mathcal{A}$ and $\mathcal{B}$, respectively. The players then move their pebbles alternatingly as we have described before. However in this case, if we obtain an infinite play, Duplicator does not immediately win. The winning condition for Duplicator is also defined with respect to the acceptance of $\mathcal{A}$ and $\mathcal{B}$. There are several possibilities to define this.

- If we consider the *direct* winning condition as in [DHWT92], Duplicator wins iff for each round $i > 0$, whenever Spoiler visits an accepting state in $\mathcal{A}$ then she also visits an accepting state $\mathcal{B}$.
- If we consider the *delay* winning condition as in [EWS01], Duplicator wins iff for each round $i > 0$, whenever Spoiler visits an accepting state in $\mathcal{A}$, Duplicator also visits an accepting state in $\mathcal{B}$ in some round $i' \geq i$.
- If we consider the *fair* winning condition as in [HKR02], Duplicator wins iff whenever Spoiler visits an accepting state in $\mathcal{A}$ infinitely often, Duplicator also visits an accepting state in $\mathcal{B}$ infinitely often.

Note that if Duplicator wins with the direct winning condition then she also wins with the delay winning condition. Furthermore, if Duplicator wins with the delay winning condition then she also wins with the fair winning condition. The converses of these two properties however do not hold.

Example 2.5.2. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$.



In this case, Duplicator can play such that the play goes on for infinitely many rounds. Note that if we consider the direct winning condition, Duplicator loses the simulation game. In the first round, SPOILER moves his pebble to an accepting state, but DUPLICATOR does not. Hence DUPLICATOR loses. However, if we consider the delay winning condition then DUPLICATOR wins. In the second round, DUPLICATOR moves her pebble to an accepting state, and after this round, SPOILER never visits an accepting state anymore. Moreover, if we consider the fair winning condition, DUPLICATOR also wins since SPOILER does not visit an accepting state infinitely often.

If we swap the automata, that is, SPOILER now controls the pebble in $\mathcal{B}$ and DUPLICATOR in $\mathcal{A}$, then DUPLICATOR loses if we consider the direct and delay winning conditions. This is because SPOILER sees an accepting state in the second round, but started from this round, DUPLICATOR does not see any accepting state. DUPLICATOR however still wins the simulation game if we consider the fair winning condition since SPOILER does not see an accepting state infinitely often.

Simulation with the direct, delay, or fair winning conditions can be used to approximate language inclusion, in the sense that, language inclusion holds if DUPLICATOR wins the simulation game. However, simulation with the fair winning condition provides a better approximation. There might be a case where language inclusion holds and DUPLICATOR loses the simulation game with the direct or delay winning condition, but wins with the fair winning condition.

Example 2.5.3. Consider again the NBA $\mathcal{A}$, $\mathcal{B}$ in Example 2.5.2. We have $L(\mathcal{A}) = L(\mathcal{B}) = \emptyset$, and hence we have language inclusion $L(\mathcal{B}) \subseteq L(\mathcal{A})$. In the game where SPOILER controls the pebble in $\mathcal{B}$ and DUPLICATOR in $\mathcal{A}$, we have seen that DUPLICATOR loses if we consider direct or delay condition, but wins with the fair condition.

The direct and delay winning conditions are simply too strong for approximating language inclusion. Through out this work we will only consider simulation with the fair winning condition. We formally define the simulation game that we will refer throughout this work in the framework of two-player game as in Section 2.4.

Definition 2.5.4 ([HKR02]). Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ. The *(fair) simulation game* is $\mathcal{G}(\mathcal{A},\mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where $V = V_0 \cup V_1$, SPOILER's and DUPLICATOR's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times Q^{\mathcal{B}} \times \{\mathsf{S}\},$$
$$V_0 = Q^{\mathcal{A}} \times \Sigma \times Q^{\mathcal{B}} \times \{\mathsf{D}\},$$

the edge relation E is defined as

$$(p, q, \mathsf{S}) \to (p', a, q, \mathsf{D}) \text{ in } E \qquad \text{iff} \qquad p \xrightarrow{a} p',$$

$$(p, a, q, \mathsf{D}) \to (p, q', \mathsf{S}) \text{ in } E \qquad \text{iff} \qquad q \xrightarrow{a} q',$$

the initial configuration is $v_0 = (p_0, q_0, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and an infinite play $v_0 v_1 v_1 \ldots \in \mathsf{Win}$ iff either there exist infinitely many i such that $v_i = (p, q, \mathsf{S})$ with $q \in F^{\mathcal{B}}$ or there are only finitely many i such that $v_i = (p, q, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. We write $\mathcal{A} \sqsubseteq \mathcal{B}$ if DUPLICATOR wins $\mathcal{G}(\mathcal{A},\mathcal{B})$.

If DUPLICATOR wins the simulation game $\mathcal{G}(\mathcal{A},\mathcal{B})$ then we have language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Every accepting word that can be produced by SPOILER in $\mathcal{A}$ can be mimicked by DUPLICATOR in $\mathcal{B}$. We formally show this as follows.

Proposition 2.5.5. *If $\mathcal{A} \sqsubseteq \mathcal{B}$ then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

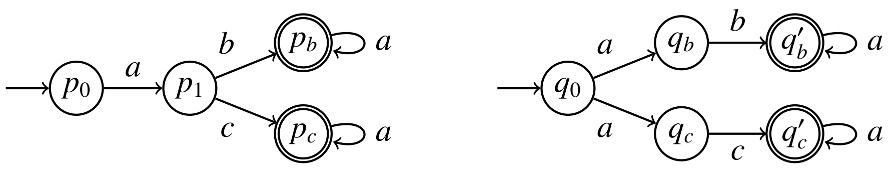
Proof. Suppose Duplicator wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Let $w = a_1 a_2 \ldots \in L(\mathcal{A})$ be some arbitrary accepting word in $\mathcal{A}$ and $\rho = p_0 a_1 p_1 \ldots$ an accepting run over w in $\mathcal{A}$. Consider the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$ where Spoiler plays ρ and Duplicator plays according to the winning strategy. Suppose we obtain a play $v'_0 v_1 v'_1 \ldots$ in $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Hence $v'_0 = (p_0, q_0, \mathsf{S})$ is the initial configuration of $\mathcal{G}(\mathcal{A}, \mathcal{B})$ and for all $i > 0$, we have $v_i = (p_i, a_i, q_{i-1}, \mathsf{D})$ and $v'_i = (p_i, q_i, \mathsf{S})$ where q_{i-1}, $q_i \in Q^{\mathcal{B}}$. The state q_0 is the initial state of $\mathcal{B}$ and by definition of the edge relation in $\mathcal{G}(\mathcal{A}, \mathcal{B})$, we have

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \tag{2.1}$$

Moreover, since Duplicator wins the play $v'_0 v_1 v'_1 \ldots$, there are infinitely many i such that $q_i \in F^B$. Hence the run in (2.1) is accepting and $w \in L(\mathcal{B})$. Since we consider an arbitrary w, we have $L(\mathcal{A}) \subseteq L(\mathcal{B})$. □

The converse of this proposition, however, does not hold. We can simply construct two NBA where language inclusion holds, but simulation does not.

Example 2.5.6. Consider again the following two NBA $\mathcal{A}$ (left) and $\mathcal{B}$ (right) over the alphabet $\Sigma = \{a, b, c\}$ that we have presented in Figure 1.1. For convenience, we present them again as follows.



In this case, we have $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and indeed $L(\mathcal{A}) = L(\mathcal{B})$. However, Spoiler wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$ with the following winning strategy. Initially, he moves his pebble from p_0 to p_1 by reading a, i.e. he proceeds from the initial configuration (p_0, q_0, S) to $(p_1, a, q_0, \mathsf{D})$. Duplicator then will move her pebble from q_0 to some q_x where $x \in \{b, c\}$ and proceed to the configuration (p_1, q_x, S). From such a configuration, Spoiler reads $\overline{x} \in \{b, c\} \setminus \{x\}$ by moving his pebble from p_1 to $p_{\overline{x}}$. Hence we reach the configuration

$$(p_{\overline{x}}, \overline{x}, q_x, \mathsf{D}). \tag{2.2}$$

Such a configuration does not have a valid successor since from the state q_x there is no $\overline{x}$-transition. Hence Duplicator gets stuck in the second round and loses the simulation game $\mathcal{G}(\mathcal{A}, \mathcal{B})$.

The simulation game that is defined in Definition 2.5.4 can be seen as a parity game. Duplicator corresponds to player 0 and Spoiler to player 1. It is not hard to see that the winning condition of the simulation game can be expressed as a parity condition.

Proposition 2.5.7. *For any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ, there is a parity game $\mathcal{G}$ with priorities* 0, 1, *and* 2 *where the numbers of nodes and edges are respectively*

$$|V| = O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|),$$
$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|)$$

such that Duplicator *wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

Proof. Let $G = (V, V_0, V_1, E)$ be the configuration graph of $\mathcal{G}(\mathcal{A}, \mathcal{B})$ and v_0 the initial configuration of $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Consider the parity game $\mathcal{G} = (G, v_0, \Omega)$ where Ω is a priority function that mimics the winning condition of Duplicator in the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$, i.e.

$$\Omega(v) = \begin{cases} 2, & \text{if } v = (p, q, \mathsf{S}) \text{ and } q \in F^{\mathcal{B}} \\ 1, & \text{if } v = (p, a, q, \mathsf{D}) \text{ and } p \in F^{\mathcal{A}} \\ 0, & \text{otherwise.} \end{cases}$$

Suppose Duplicator wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$. In the parity game $\mathcal{G}$, player 0 plays according to the winning strategy for Duplicator in $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Suppose we obtain a play $\pi = v_0 v_1 \ldots$. Since Duplicator wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$, either there exist infinitely many i such that $v_i = (p, q, \mathsf{S})$ with $q \in F^{\mathcal{B}}$ or there are only finitely many i such that $v_i = (p, a, q, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. In the first case, the highest priority that is seen infinitely often is 2 and in the second case, it is 0. Hence player 0 wins $\mathcal{G}$. We can also show the other direction similarly.

The size of the constructed parity game $\mathcal{G}$ is the same as the size of the configuration graph of $\mathcal{G}(\mathcal{A}, \mathcal{B})$. It has at most $|\mathcal{A}| \cdot |\mathcal{B}| \cdot (|\Sigma| + 1)$ many nodes and $|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|$ many edges. □

Since by Proposition 2.4.2, deciding the winner of a parity game with n nodes, m edges, and priorities 0, 1, and 2 can be done in time $O(nm)$, we have the following proposition.

Proposition 2.5.8. *For any two NBA $\mathcal{A}$, $\mathcal{B}$, deciding $\mathcal{A} \sqsubseteq \mathcal{B}$ is in time $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^2)$.*

This shows that deciding simulation can be done in polynomial time. We have a polynomial approximation for language inclusion. Such a cheap approximation, however, comes at price. We have seen in Example 2.5.6 that simulation is too weak to show language inclusion. There are even simple NBA $\mathcal{A}$, $\mathcal{B}$ in which their language inclusion cannot be shown by simulation. One reasonable question then is to ask whether we can extend simulation such that it can capture more language inclusion. In the next section, we give some extended simulations that are available in the literature.

2.6 Extended Simulations

One simple reason why the standard fair simulation fails to show language inclusion is because Duplicator's power is very limited. Duplicator only sees the accepting run that is formed by Spoiler in $\mathcal{A}$ step by step, but has to guess the corresponding run in $\mathcal{B}$ correctly. This is in contrast to language inclusion in which one can see the whole accepting run in $\mathcal{A}$ and then decide whether there is a corresponding run in $\mathcal{B}$. Hence the typical extension of simulation is to modify the game framework such that either Duplicator has more power or Spoiler has less power.

2.6.1 Static Multi-Letter Simulation

In the static multi-letter simulation, the game framework is modified such that Spoiler is weaker than in the standard fair simulation game. He is forced to read more than one letter in each round. More precisely, in the *static k-letter simulation*, where $k > 0$, in each round, Spoiler moves the pebble k steps over some word $a_1 \ldots a_k$ and Duplicator tries to mimic this move by moving the pebble k steps over the same word. The winning condition

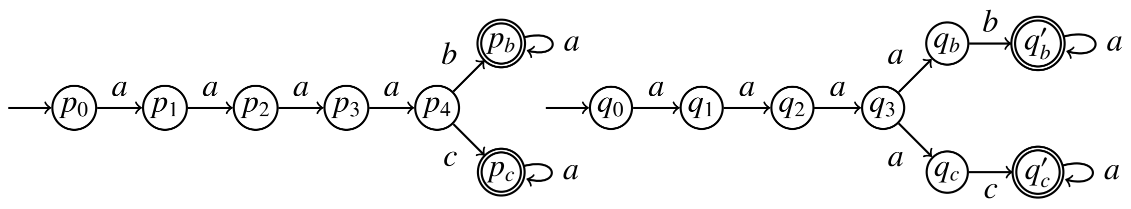


Figure 2.2: NBA $\mathcal{A}$, $\mathcal{B}$ in which $\mathcal{A} \not\sqsubseteq^2_{\mathsf{Stat}} \mathcal{B}$.

is the same as in the standard fair simulation. Duplicator wins iff Spoiler eventually gets stuck or whenever Spoiler forms an accepting run, Duplicator also forms an accepting run.

Definition 2.6.1 ([HLL13]). Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ and $k > 0$. The *static k-letter simulation game* is $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{S}\},$$
$$V_0 = Q^{\mathcal{A}} \times \Sigma^k \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

$$(p, q, b, \mathsf{S}) \to (p', w, q, b', \mathsf{D}) \text{ in } E \quad \text{iff} \quad p \xrightarrow{w} p' \text{ and } b' = \begin{cases} \top & \text{if } p \xrightarrow[F]{w} p' \\ \bot & \text{otherwise,} \end{cases}$$

$$(p, w, q, b, \mathsf{D}) \to (p, q', b', \mathsf{S}) \text{ in } E \quad \text{iff} \quad q \xrightarrow{w} q' \text{ and } b' = \begin{cases} \top & \text{if } q \xrightarrow[F]{w} q' \\ \bot & \text{otherwise,} \end{cases}$$

the initial configuration is $v_0 = (p_0, q_0, \bot, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and an infinite play $v_0 v_1 \ldots \in \mathsf{Win}$ iff either there exist infinitely many i such that $v_i = (p_i, q_i, \top, \mathsf{S})$ or there are only finitely many i such that $v_i = (p_i, q_i, \top, \mathsf{D})$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ if Duplicator wins $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$.

Example 2.6.2. Consider again the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 2.5.6. We have $\mathcal{A} \sqsubseteq^2_{\mathsf{Stat}} \mathcal{B}$. In the first round of the game $\mathcal{G}^2_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$, initially Spoiler will move his pebble two steps from p_0 to some state p_x where $x \in \{b, c\}$ by reading ax, i.e. he proceeds from the initial configuration $(p_0, q_0, \bot, \mathsf{S})$ to $(p_x, ax, q_0, \top, \mathsf{D})$. Duplicator then responds to this by moving her pebble from q_0 to q_x, i.e. she proceeds to

$$(p_x, q_x, \top, \mathsf{S}). \tag{2.3}$$

From such a configuration, after Spoiler reads aa by looping in p_x and proceeding to $(p_x, aa, q_x, \top, \mathsf{D})$, Duplicator reads aa by looping in q_x and proceeding back to (2.3). Duplicator repeats this procedure indefinitely. Hence Duplicator visits an accepting state infinitely often. She forms an accepting run. We obtain a play $v_0 v_1 \ldots$ where there are infinitely many i such that $v_i = (p, q, \top, \mathsf{S})$. Duplicator wins the game $\mathcal{G}^2_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$.

Example 2.6.3. Consider the two NBA $\mathcal{A}, \mathcal{B}$ in Figure 2.2. We have $\mathcal{A} \not\sqsubseteq^2_{\mathsf{Stat}} \mathcal{B}$. Spoiler wins the game $\mathcal{G}^2_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$ with the following winning strategy. Initially, he reads aa and hence the play proceeds from the initial configuration $(p_0, \epsilon, q_0, \perp, \mathsf{S})$ to $(p_2, aa, q_0, \perp, \mathsf{D})$. Duplicator does not have any choice except to read aa by going to q_2 and reaching the configuration $(p_2, q_2, \perp, \mathsf{S})$. From this configuration, Spoiler again reads aa and proceeds to $(p_4, aa, q_2, \perp, \mathsf{D})$. Duplicator again does not have any choice except to read aa by going to some state q_x where $x \in \{b, c\}$. The play then reaches the configuration $(p_4, q_x, \perp, \mathsf{S})$. In the next round, Spoiler reads $\overline{x}a$ where $\overline{x} \in \{b, c\} \setminus \{x\}$ and proceeds to the configuration $(p_{\overline{x}}, \overline{x}a, q_x, \top, \mathsf{D})$. Such a configuration, however, does not have a valid successor since there is no $\overline{x}$-transition from q_x. Hence in the third round, Duplicator gets stuck and loses the play. Spoiler wins the game $\mathcal{G}^2_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$.

Note that if $k = 1$, the static k-letter simulation game is equivalent to the standard fair simulation game. Spoiler and Duplicator only move one step in each round. Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ iff she wins the static k-letter simulation game $\mathcal{G}^1_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$.

Proposition 2.6.4 ([HLL13])**.** $\sqsubseteq^1_{\mathsf{Stat}} = \sqsubseteq$.

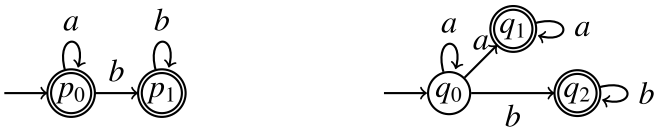
Similarly to the ordinary fair simulation, static k-letter simulation also approximates language inclusion. If we have $\mathcal{A} \sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ for some $k \in \mathbb{N}$ then we also have language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. The reason is also similar to the of the standard fair simulation. Every accepted word that can be produced by Spoiler in $\mathcal{A}$ can be mimicked by Duplicator in $\mathcal{B}$ by playing according to the winning strategy in $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$.

Proposition 2.6.5 ([HLL13])**.** *For any two NBA $\mathcal{A}$, $\mathcal{B}$, if there is $k \in \mathbb{N}$ such that $\mathcal{A} \sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

Note that the reverse direction of this theorem does not hold. Similarly to ordinary simulation, language inclusion does not imply static multi-letter simulation. There are some cases in which language inclusion holds, but static k-letter simulation does not hold.

Corollary 2.6.6. *There are two NBA $\mathcal{A}$, $\mathcal{B}$ such that $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ for all $k > 0$.*

Proof. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$.



We have $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ for any $k > 0$. Spoiler has a winning strategy in the game $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$. He reads a^k indefinitely by looping in p_0 until Duplicator moves to q_1, i.e. from any configuration (p_0, q, b, S) where $q \neq q_1$ and $b \in \{\perp, \top\}$, Spoiler proceeds to $(p_0, a^k, q, \top, \mathsf{D})$ by reading a^k. However if Duplicator eventually moves to q_1 then he moves to p_1, i.e. from the configuration $(p_0, q_1, \top, \mathsf{S})$, Spoiler proceeds to

$$(p_1, b^k, q_1, \top, \mathsf{D}) \tag{2.4}$$

by reading b^k.

Now suppose Duplicator never moves to q_1. Hence she does not form an accepting run. We obtain a play $v'_0 v_1 v'_1 \ldots$ where for all $i > 0$, $v_i = (p_0, a^k, q_0, \top, \mathsf{D})$ and $v'_i = (p_0, q_0, \perp, \mathsf{S})$. Hence Spoiler wins. However, if Duplicator eventually moves to q_1, then in the next round, the play proceeds to (2.4). Since there is no b-transition from q_1, such a configuration does not have a valid successor. DUPLICATOR gets stuck and hence loses the play. SPOILER wins the game $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$ for all $k > 0$. □

We can also see the static k-letter simulation as a parity game similarly as in the case of the fair simulation. We can consider the same priority function as in (3.6). However the size of the configuration graph of $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$ is slightly bigger than the one of $\mathcal{G}(\mathcal{A},\mathcal{B})$, Recall from Definition 2.6.1 that SPOILER's and DUPLICATOR's configurations in $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$ are of the form (p, q, b, S) and (p, w, q, b, D) where $p \in Q^{\mathcal{A}}$, $q \in Q^{\mathcal{B}}$, $w \in \Sigma^k$, and $b \in \{\bot, \top\}$. Hence there are $2 \cdot |\mathcal{A}| \cdot |\mathcal{B}| \cdot (|\Sigma|^k + 1)$ many configurations. Moreover, the edges in the configuration graph of $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$ are of the form

$$(p, q, b, \mathsf{S}) \to (p', w, q, b', \mathsf{D}) \text{ or}$$
$$(p, w, q, b, \mathsf{D}) \to (p, q', b', \mathsf{S}).$$

where $p, p' \in Q^{\mathcal{A}}$, $q, q' \in Q^{\mathcal{B}}$, $w \in \Sigma^k$, and $b, b' \in \{\bot, \top\}$. Hence the number of edges in the corresponding parity game is at most $4 \cdot |\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|^k$.

Proposition 2.6.7 ([HLL13]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ and $k > 0$, there is a parity game $\mathcal{G}$ with priorities 0, 1, and 2 where the numbers of nodes and edges are*

$$|V| = O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|^k),$$
$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot (|\Sigma|^k)$$

such that DUPLICATOR *wins $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

By Proposition 2.4.2, since deciding the winner of a parity game with n nodes, m edges, and priorities 0, 1, and 2 can be done in time $O(nm)$, we have the following proposition.

Proposition 2.6.8 ([HLL13]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ, deciding whether $\mathcal{A} \sqsubseteq^k_{\mathsf{Stat}} \mathcal{B}$ is in time $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{2k})$.*

2.6.2 Dynamic Multi-Letter Simulation

Another extension of simulation game that is introduced in [HLL13] is the *dynamic k-letter simulation* game. This simulation is also independently introduced in [CM13]. Intuitively the game proceeds as follows. In the beginning of each round $i > 0$, DUPLICATOR first chooses some $\ell_i \leq k$. SPOILER then moves his pebble ℓ_i steps by reading some word w of length ℓ_i and DUPLICATOR tries to mimic this move by moving her pebble ℓ_i steps by reading the same word w. The winning condition of the game does not differ from the static multi-letter or the standard fair simulation. If one of the players gets stuck then the opponent wins, otherwise DUPLICATOR wins iff whenever SPOILER forms an accepting run, she also forms one. Note that the static k-letter simulation is indeed a special case of the dynamic one where DUPLICATOR always chooses $\ell_i = k$ for each round.

Let us denote by $\Sigma^{\leq k}$ the set of finite words over Σ of length at most k. The game is formally defined as follows.

Definition 2.6.9 ([CM13, HLL13]). Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ and $k > 0$. The dynamic k-letter simulation game is $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where SPOILER's and DUPLICATOR's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times Q^{\mathcal{B}} \times \{1, \ldots, k\} \times \{\top, \bot\} \times \{\mathsf{S}\},$$

$$V_0 = Q^{\mathcal{A}} \times \Sigma^{\leq k} \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

$$(p, q, \ell, b, \mathsf{S}) \to (p', w, q, b', \mathsf{D}) \text{ in } E \qquad \text{iff} \qquad \begin{array}{l} w \in \Sigma^{\ell},\ p \xrightarrow{w} p',\ \text{and} \\ b' = \begin{cases} \top & \text{if } p \xrightarrow[F]{w} p' \\ \bot & \text{otherwise,} \end{cases} \end{array}$$

$$(p, w, q, b, \mathsf{D}) \to (p, q', \ell, b', \mathsf{S}) \text{ in } E \qquad \text{iff} \qquad \begin{array}{l} q \xrightarrow{w} q', \ell \leq k,\ \text{and} \\ b' = \begin{cases} \top & \text{if } q \xrightarrow[F]{w} q' \\ \bot & \text{otherwise,} \end{cases} \end{array}$$

the initial configuration is $v_0 = (p_0, \epsilon, q_0, \bot, \mathsf{D})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}, \mathcal{B}$, and an infinite play $v_0v_1 \ldots \in \mathsf{Win}$ iff either there exist infinitely many i such that $v_i = (p_i, w_i, q_i, \top, \mathsf{S})$ or there are only finitely many i such that $v_i = (p_i, q_i, \ell_i, \top, \mathsf{D})$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ if Duplicator wins $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$.

Example 2.6.10. Consider again the two NBA $\mathcal{A}, \mathcal{B}$ in Figure 2.2. We have $\mathcal{A} \sqsubseteq^2_{\mathsf{Dyn}} \mathcal{B}$. Duplicator wins the game $\mathcal{G}^2_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ with the following winning strategy. Initially she asks Spoiler to move the pebble two steps, i.e. she proceeds from the initial configuration $(p_0, \epsilon, q_0, \bot, \mathsf{D})$ to $(p_0, q_0, 2, \bot, \mathsf{S})$. Spoiler then does not have any choice except to read aa and reach the configuration $(p_2, aa, q_0, \bot, \mathsf{D})$. From this configuration, Duplicator reads aa, moves her pebble from q_0 to q_2, and asks Spoiler to move her pebble one step for the next round, i.e. Duplicator proceeds to $(p_2, q_2, 1, \bot, \mathsf{S})$. Spoiler again does not have any choice except to read a and continue to the configuration $(p_3, a, q_2, \bot, \mathsf{D})$. Duplicator reads a, moves her pebble from q_2 to q_3, and asks Spoiler to move two steps for the next round. She proceeds to $(p_3, q_3, 2, \bot, \mathsf{S})$. From this configuration, Spoiler might either read ab or ac, i.e. he proceeds to some configuration $(p_x, ax, q_3, \top, \mathsf{D})$ where $x \in \{b, c\}$. Duplicator then responds to this by reading ax, moving her pebble from q_3 to q_x, and asking Spoiler to move one step for the next round. She proceeds to

$$(p_x, q_x, 1, \top, \mathsf{S}). \tag{2.5}$$

From this configuration, whenever Spoiler continues to $(p_x, a, q_x, \top, \mathsf{D})$, Duplicator proceeds back to (2.5). She repeats this procedure indefinitely. Hence Duplicator forms an accepting run. We obtain a play $v_0v_1 \ldots$ in which there exist infinitely many i such that $v_i = (p, q, \ell, \top, \mathsf{S})$. By definition, Duplicator wins the play. She wins the game $\mathcal{G}^2_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$.

Similarly to static k-letter simulation, in the case where $k = 1$, dynamic k-letter simulation is equivalent to the fair simulation. Duplicator wins the game $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ iff she wins the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$.

Proposition 2.6.11 ([CM13, HLL13]). $\sqsubseteq^1_{\mathsf{Dyn}} = \sqsubseteq$.

It is not hard to see that if Duplicator wins the static k-letter simulation game, she also wins the dynamic k-letter simulation game. She simply chooses $\ell_i = k$ in every round $i > 0$ and moves exactly as in the static k-letter simulation game.

Proposition 2.6.12 ([HLL13]). *For all $k > 0$, $\sqsubseteq^k_{\mathsf{Stat}} \subseteq \sqsubseteq^k_{\mathsf{Dyn}}$.*

The other direction, however, does not hold. Duplicator might lose the static k-letter simulation game even though she wins the dynamic one. Consider again the two NBA $\mathcal{A}$ and $\mathcal{B}$ in Figure 2.2. We have seen in Example 2.6.10 that Duplicator wins the game $\mathcal{G}^2_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$. However, she loses the game $\mathcal{G}^2_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$ as we have seen in Example 2.6.3.

However, similarly to the static multi-letter simulation, dynamic multi-letter simulation also approximates language inclusion. If there exists $k > 0$ such that $\mathcal{A} \sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ then we have language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. The reason is similar to the case of the fair simulation and the static k-letter simulation. If Duplicator wins $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ for some $k > 0$ then for every accepting word that can be produced by Spoiler in $\mathcal{A}$, Duplicator can mimic it in $\mathcal{B}$ by playing according to the winning strategy in $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$.

Proposition 2.6.13 ([CM13, HLL13]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ if there is $k > 0$ such that $\mathcal{A} \sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

The approximation of language inclusion with dynamic k-letter simulation is not complete. There are some cases in which language inclusion holds, but dynamic k-letter simulation does not hold for any $k > 0$. We can consider the same NBA $\mathcal{A}$, $\mathcal{B}$ as in the static case.

Proposition 2.6.14. *There are two NBA $\mathcal{A}$, $\mathcal{B}$ such that $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ for all $k > 0$.*

Proof. Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ as in the proof of Proposition 2.6.6. We have $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ for any $k > 0$. In the game $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$, Spoiler wins by considering a similar winning strategy as in the proof of Proposition 2.6.6, i.e. he reads a^ℓ for any length ℓ that is chosen by Duplicator. He reads a^ℓ by looping in p_0 until Duplicator moves to q_1. Formally, from any configuration $(p_0, q, \ell, b, \mathsf{S})$ where $\ell \leq k$ and $b \in \{\bot, \top\}$, if $q \neq q_1$ then Spoiler proceeds to $(p_0, a^\ell, q, \top, \mathsf{D})$, and if $q = q_1$ then he proceeds to $(p_1, b^\ell, q_1, \top, \mathsf{S})$. We can show similarly as in the proof of Proposition 2.6.6 that Spoiler wins $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ by using such a strategy. □

We can also see the dynamic k-letter simulation game as a parity game in the same way as in the case of the static k-letter simulation. The size of the parity game that corresponds to the dynamic k-letter simulation game is slightly bigger than the static one. Recall from Definition 2.6.9 that Spoiler's and Duplicator's configurations in $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ are of the forms $(p, q, \ell, b, \mathsf{S})$ and (p, w, q, b, D) where $p \in Q^{\mathcal{A}}$, $q \in Q^{\mathcal{B}}$, $\ell \in \{1, \ldots, k\}$, $w \in \Sigma^{\leq k}$, and $b \in \{\bot, \top\}$. Since $|\Sigma^{\leq k}| = |\Sigma^1| + \ldots + |\Sigma^k| = \mathcal{O}(|\Sigma|^{k+1})$, we will obtain a parity game with $2 \cdot |\mathcal{A}| \cdot |\mathcal{B}| \cdot (k + |\Sigma|^{k+1})$ nodes. Moreover, the edges in the configuration graph of $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ are of the form

$$(p, q, \ell, b, \mathsf{S}) \to (p', w, q, b', \mathsf{D}) \text{ or}$$
$$(p, w, q, b, \mathsf{D}) \to (p, q', \ell, b', \mathsf{S}).$$

where $p, p' \in Q^{\mathcal{A}}$, $q, q' \in Q^{\mathcal{B}}$, $w \in \Sigma^k$, $\ell \in \{1, \ldots, k\}$, and $b, b' \in \{\bot, \top\}$. Hence the number of edges in the corresponding parity game is at most $4k \cdot |\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{k+1}$.

Proposition 2.6.15 ([HLL13]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ and $k > 0$, there is a parity game $\mathcal{G}$ with priorities 0, 1, and 2 where the number of nodes and edges are respectively*

$$|V| = \mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|^{k+1}),$$

$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{k+1})$$

such that Duplicator *wins* $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A}, \mathcal{B})$ *iff player 0 wins the parity game* $\mathcal{G}$.

Since deciding the winner of a parity game with $|V|$ many nodes, $|E|$ many edges, and priorities 0, 1, and 2 can be done in time $O(|V| \cdot |E|)$, we have the following proposition.

Proposition 2.6.16 ([CM13, HLL13])**.** *For any two NBA* $\mathcal{A}$, $\mathcal{B}$ *over* Σ *and* $k \in \mathbb{N}$, *deciding* $\mathcal{A} \sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$ *is in time* $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{2k+2})$.

2.6.3 Multi-Pebble Simulation

Multi-pebble simulation is a bit different from the static and dynamic multi-letter simulations. The players move alternatingly one step in each round as in the ordinary simulation game. However in this case, Duplicator is allowed to control several pebbles [Ete02]. In the k-pebble simulation game, where $k > 0$, Duplicator controls k pebbles. Initially, Spoiler's pebble is placed in the initial state of $\mathcal{A}$ and all of Duplicator's pebbles are placed in the initial state of $\mathcal{B}$. The play then proceeds as follows. In each round $i > 0$, Spoiler moves his pebble in $\mathcal{A}$ one step by reading some letter a. Duplicator chooses some a-successors of the states which are currently occupied by her pebbles in $\mathcal{B}$. She then moves her pebbles to the chosen successors by reading a. Note that there might be a case where some of Duplicator's pebbles cannot be moved, e.g. suppose Duplicator controls 2 pebbles which are currently in the states q_1 and q_2, q_1 has an a-successor, but q_2 does not. In such a case, Duplicator drops the pebbles that cannot be moved, duplicates the remaining ones so she has k pebbles again, and then moves them to the chosen successors. Duplicator wins an infinite play iff either Spoiler's pebble does not form an accepting run or all of her pebbles that survive infinitely many rounds see some accepting state infinitely often.

Let us denote by $\mathcal{P}_{\leq k}(S)$ the set of all subsets of S of size at most k. Hence if $|S| = n$ then $|\mathcal{P}_{\leq k}(S)| \leq (n+1)^k$. The game is formally defined as follows.

Definition 2.6.17 ([Ete02])**.** Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ and $k > 0$. The *k-pebble simulation game* is $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are respectively

$$V_1 = Q_{\mathcal{A}} \times (\mathcal{P}_{\leq k}(Q^{\mathcal{B}}) \setminus \{\emptyset\}) \times \mathcal{P}_{\leq k}(Q^{\mathcal{B}}) \times \{\mathsf{S}\},$$
$$V_0 = Q_{\mathcal{A}} \times \Sigma \times (\mathcal{P}_{\leq k}(Q^{\mathcal{B}}) \setminus \{\emptyset\}) \times \mathcal{P}_{\leq k}(Q^{\mathcal{B}}) \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

$(p, Q, R, \mathsf{S}) \to (p', a, Q, R, \mathsf{D})$ in E iff $p \xrightarrow{a} p'$

$(p, a, Q, R, \mathsf{D}) \to (p, Q', R', \mathsf{S})$ in E iff
$\forall q' \in Q' \; \exists q \in Q$ such that $q \xrightarrow{a} q'$ and
$R' = \{r' \in Q' \backslash F^{\mathcal{B}} \mid \exists r \in R : r \xrightarrow{a} r'\}$ if $R \neq \emptyset$,
$R' = Q' \backslash F^{\mathcal{B}}$ if $R = \emptyset$,

the initial configuration is $v_0 = (p_0, \{q_0\}, \emptyset, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and an infinite play $v_0 v_1 \ldots \in \mathsf{Win}$ iff either there exist infinitely many i such that $v_i = (p, Q, \emptyset, \mathsf{S})$ or there are only finitely many i such that $v_i = (p, Q, R, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ if Duplicator wins $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$.

The first and the third components in the configuration tell us the position of Spoiler's and Duplicator's pebbles. Since Duplicator uses k many pebbles, instead of one state, the third component is a set of at most k many states. The third component is used to define the winning condition of Duplicator. It remembers which pebbles that have not seen an accepting state since the last reset. Each time Duplicator moves her pebbles, we also update this component. Once it becomes empty then we reset. We put back again the current position of all Duplicator's pebbles. Hence if we visit infinitely often configurations in which the third component is empty then all Duplicator's pebbles that survive infinitely many rounds see an accepting state infinitely often. The second component in Duplicator's configuration is the letter that has been read by Spoiler and has to be read by Duplicator, whereas the last component, as in the other simulation games we have described before, tells us which player that has the next turn.

Example 2.6.18. Consider again the NBA $\mathcal{A}$, $\mathcal{B}$ from Example 2.5.6. We have $\mathcal{A} \sqsubseteq^2_{\mathsf{Peb}} \mathcal{B}$. In the game $\mathcal{G}^2_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$, Duplicator has the following winning strategy. In the first round, after Spoiler moves his pebble from p_0 to p_1 by reading a, i.e. proceeds from the initial configuration $(p_0, \{q_0\}, \emptyset, \mathsf{S})$ to $(p_1, a, \{q_0\}, \emptyset, \mathsf{D})$, Duplicator moves her two pebbles from q_0 each to q_b and q_c. She proceeds to $(p_1, \{q_b, q_c\}, \{q_b, q_c\}, \mathsf{S})$. In the second round, after Spoiler moves his pebble from p_1 to some p_x where $x \in \{b, c\}$, by reading x, we reach the configuration $(p_x, x, \{q_b, q_c\}, \{q_b, q_c\}, \mathsf{D})$. Duplicator then drops the pebble in $q_{\overline{x}}$ where $\overline{x} \in \{b, c\} \setminus \{x\}$, duplicates the one at q_x once, and moves them to q'_x, i.e. she continues to $(p_x, \{q'_x\}, \emptyset, \mathsf{S})$. From this configuration, for the rest of the play, whenever Spoiler reads a and proceeds to $(p_x, a, \{q'_x\}, \emptyset, \mathsf{D})$, Duplicator also reads a by moving her pebbles through the loop in q_x and proceeding back to $(p_x, \{q'_x\}, \emptyset, \mathsf{S})$. All Duplicator pebbles form an accepting run. We obtain a play $v_0 v_1 \ldots$ in which there are infinitely many i such that $v = (p, Q, \emptyset, \mathsf{S})$. Hence Duplicator wins the play. She wins the game $\mathcal{G}^2_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$.

Note that if $k = 1$ then it is not hard to see that the k-pebble game is equivalent to the fair simulation game since Duplicator only controls one pebble. Duplicator wins the game $\mathcal{G}^1_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ iff she wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$.

Proposition 2.6.19 ([Ete02])**.** $\sqsubseteq^1_{\mathsf{Peb}} = \sqsubseteq$.

Intuitively, the multi-pebble game extends the fair simulation game by giving Duplicator more power by controlling k many pebbles. Such a simulation also approximates language inclusion in the same sense as the static and dynamic multi-letter simulations.

Proposition 2.6.20 ([Ete02])**.** *For any two NBA $\mathcal{A}$, $\mathcal{B}$, if there is $k > 0$ such that $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

The reverse direction of this proposition, however, does not hold. Similarly to static and dynamic multi-letter simulations, the approximation for language inclusion with multi-pebble simulation is also not complete. There is an example of a language inclusion which cannot be shown by multi-pebble simulation. We can even consider the same pair of NBA as in the case of static and dynamic multi-letter simulations.

Corollary 2.6.21. *There are two NBA $\mathcal{A}$, $\mathcal{B}$ such that $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ for all $k > 0$.*

Proof. Consider again the two NBA $\mathcal{A}, \mathcal{B}$ in the proof of Proposition 2.6.6. We have $L(\mathcal{A}) = L(\mathcal{B})$ but $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ for any $k \in \mathbb{N}$. In the game $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$, Spoiler has a winning strategy as follows. She reads a indefinitely by looping in p_0 until Duplicator moves all her pebbles from the initial state to q_1, i.e. from any configuration (p_0, Q, R, S) where $Q \neq \{q_1\}$, Spoiler proceeds to $(p_0, a, Q, R, \mathsf{D})$. However if Duplicator eventually moves all her pebbles to q_1 then Spoiler moves to p_1 by reading b, i.e. from the configuration $(p_0, \{q_1\}, \emptyset, \mathsf{S})$, Spoiler proceeds to

$$(p_1, b, \{q_1\}, \emptyset, \mathsf{D}). \tag{2.6}$$

Now suppose Duplicator eventually moves all her pebbles from the initial state to q_1. In the next round, the play then proceeds to the configuration (2.6). Such a configuration does not have a valid successor since there is no b-transition from q_1. Hence Duplicator loses the play. In the case where Duplicator never moves all of her pebbles from the initial state to q_1 then one of her pebbles, namely the one that is in q_0, does not form an accepting run. We obtain a play $v'_0 v_1 v'_1 \ldots$ in which for all $i > 0$, $v'_i = \{p_0, Q_i, R_i, \mathsf{S}\}$, $Q_i \subseteq \{q_0, q_1\}$, and $q_0 \in R_i$. Since $R_i \neq \emptyset$ for all $i > 1$, by definition, Duplicator loses the play. Spoiler wins the game $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$. □

We can see the multi-pebble simulation game $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ as a parity game where player 1 corresponds to Spoiler and player 0 to Duplicator. It is not hard to see that the winning condition of the multi-pebble simulation game can be expressed as a parity condition. We can simply consider a parity game $\mathcal{G}$ that is played in the same configuration graph as $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ with a priority function that assigns priority 2 to any configuration (p, Q, R, S) with $R = \emptyset$, priority 1 to any configuration (p, a, Q, R, D) with $p \in F^{\mathcal{A}}$, and priority 0 to other configurations. In this way, a play in the parity game $\mathcal{G}$ is won by player 0 iff it is won by Duplicator in the multi-pebble simulation game $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$. Moreover, by the definition of $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$, it is not hard to see that the configuration graph of $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ has at most $|\mathcal{A}| \cdot (|\mathcal{B}| + 1)^{2k} \cdot (|\Sigma| + 1)$ nodes and $|\mathcal{A}|^2 \cdot (|\mathcal{B}| + 1)^{4k} \cdot |\Sigma|$ edges.

Proposition 2.6.22 ([Ete02]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ and $k \in \mathbb{N}$, there is a parity game $\mathcal{G}$ with priorities* 0, 1, *and* 2 *where the numbers of nodes and edges are respectively*

$$|V| = \mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}|^{2k} \cdot |\Sigma|),$$
$$|E| = \mathcal{O}(|\mathcal{A}|^2 \cdot |\mathcal{B}|^{4k} \cdot |\Sigma|)$$

such that Duplicator *wins $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

Since deciding the winner of such a parity game can be done in time linearly in the products of the number of its nodes and edges, we have the following property.

Proposition 2.6.23 ([Ete02]). *For any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ, deciding $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ is in time $\mathcal{O}(|\mathcal{A}|^3 \cdot |\mathcal{B}|^{6k} \cdot |\Sigma|^2)$.*

2.6.4 Delay Simulation

Delay simulation is introduced in [HKT10] and it is a bit different than the other extended simulations that have been mentioned before. Spoiler and Duplicator do not play on two automata, but on a regular language $L \subseteq (\Sigma_1 \times \Sigma_2)^{\omega}$ with respect to a function $f : \mathbb{N}^+ \to \mathbb{N}^+$ called *delay function*. In every round $i > 0$, Spoiler chooses some word w over Σ_1 of length $f(i)$ and Duplicator chooses some letter from Σ_2. The play goes on for infinitely many rounds and each of the players forms an infinite word. Spoiler forms an infinite word over Σ_1 and Duplicator over Σ_2. Duplicator wins the play iff the pair of the infinite words that are formed by Spoiler and Duplicator is in the language L. The game is formally defined as follows.

Definition 2.6.24 ([HKT10])**.** Let $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$ be a regular language over $\Sigma_1 \times \Sigma_2$ and $f : \mathbb{N}^+ \to \mathbb{N}^+$. The delay simulation game over the language L and delay function f is $\mathcal{G}^f_{\mathsf{Del}}(L) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are respectively

$$V_1 = \mathbb{N}^+ \times \Sigma_2 \cup \{\epsilon\} \times \{\mathsf{S}\}$$
$$V_0 = \mathbb{N}^+ \times \Sigma_1^* \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

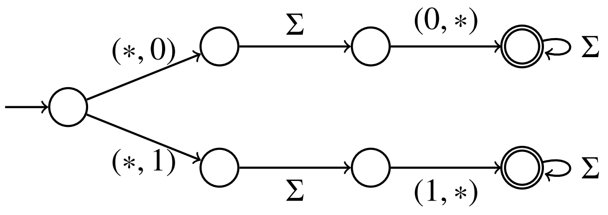
$$(i, b, \mathsf{S}) \to (i, a_1 \dots a_n, \mathsf{D}) \text{ in } E \quad \text{iff} \quad n = f(i) \text{ and } a_1 \dots a_n \in \Sigma_1^*$$
$$(i, w, \mathsf{D}) \to (i + 1, b, \mathsf{S}) \text{ in } E \quad \text{iff} \quad b \in \Sigma_2,$$

the initial configuration is $v_0 = (1, \epsilon, \mathsf{S})$, and an infinite play

$$(1, \epsilon, \mathsf{S})(1, b_1, \mathsf{D})(2, a_1 \dots a_{n_1}, \mathsf{S})(1, b_2, \mathsf{D}) \dots$$

is in Win iff $(a_1, b_1)(a_2, b_2) \dots \in L$. We say that L is solvable with f if Duplicator wins $\mathcal{G}^f_{\mathsf{Del}}(L)$.

Example 2.6.25. Consider the language L over $\Sigma = \{0, 1\} \times \{0, 1\}$ that is given by the following NBA.



By $*$ we mean any bit of $\{0, 1\}$. Hence $(a_1, b_1)(a_2, b_2) \dots \in \Sigma^\omega$ is in L iff $b_1 = a_3$. For any delay function f where $f(1) \geq 3$, the language L is solvable with f. Duplicator has a winning strategy in the game $\mathcal{G}^f_{\mathsf{Del}}(L)$. In the first round, Spoiler will choose a word $a_1 \dots a_{f(1)} \in \{0, 1\}^*$, i.e. he proceeds from the initial configuration $(1, \epsilon, \mathsf{S})$ to $(1, a_1 \dots a_{f(1)}, \mathsf{D})$. Since $f(1) \geq 3$, Duplicator then responds to this by choosing the letter a_3, i.e. she proceeds to $(1, a_3, \mathsf{S})$. Furthermore, started from the second round, no matter what Spoiler does, Duplicator simply extends her word with 0. From any configuration (i, w, D) where $i > 1$, Duplicator proceeds to $(i + 1, 0, \mathsf{S})$. In this way, Spoiler forms $a_1 a_2 \dots$ while Duplicator forms $a_3 0^\omega$. Since the word $(a_1, a_3)(a_2, 0)(a_3, 0) \dots$ is in the language L, Duplicator wins the play. She wins the game $\mathcal{G}^f_{\mathsf{Del}}(L)$.

In [HKT10], it is shown that if L is solvable with some arbitrary delay function f then it is also solvable with a special kind of delay function f' in which $f'(1) = d$ for some constant $d \in \mathbb{N}$ and $f(i) = 1$ for all $i > 1$. Such a special delay function is called *constant* delay function. If the language L can be represented by a deterministic parity automaton $\mathcal{A}$ and is solvable with some delay function f then it is also solvable with a constant delay function where the constant is doubly exponential in the size of the automaton and exponential in the number of priorities.

Proposition 2.6.26 ([HKT10]). *Given a language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$ that can be represented by a DPA with n states and m priorities, let $d = 2^{2(mn)^n+1}mn$. The language L is solvable with some f iff L is solvable with a constant delay function with constant d.*

Therefore for any regular language L that can be represented by a DPA with n states and m priorities, the problem of deciding whether there is a delay function f such that Duplicator wins delay simulation $\mathcal{G}^f_{\mathsf{Del}}(L)$ is decidable in doubly exponential time. The following is the main result in [HKT10].

Proposition 2.6.27 ([HKT10]). *For any ω-regular language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$ that can be represented by a DPA with n states and m priorities, deciding whether there exists a delay function f such that L is solvable with f is decidable in 2EXPTIME.*

Now one might wonder how delay simulation relates to the fair simulation game. A fair simulation game is indeed a special case of delay simulation game. For any two NBA $\mathcal{A}$ and $\mathcal{B}$, we can reduce the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$ to a game $\mathcal{G}^f_{\mathsf{Del}}(L)$ where f is a constant delay function with constant 1. The language L is defined over the pairs of transitions of $\mathcal{A}$ and $\mathcal{B}$ such that $(e_1, e'_1)(e_2, e'_2)\ldots \in L$ iff either $e_1e_2\ldots$ does not express a valid run in $\mathcal{A}$, it expresses a valid run in $\mathcal{A}$ but not accepting, or it expresses a valid accepting run in $\mathcal{A}$ but $e'_1e'_2\ldots$ expresses a valid accepting run in $\mathcal{B}$ over the same word. In the game $\mathcal{G}^f_{\mathsf{Del}}(L)$, Spoiler has to form a valid accepting run in $\mathcal{A}$ since otherwise he loses, and Duplicator has to mimic it in $\mathcal{B}$ stepwise by forming a corresponding accepting run. If Duplicator fails to do so then the players form a word that is not in L, and she loses the play. We show this formally as follows.

Lemma 2.6.28. *For any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ, let $L_{\mathcal{A},\mathcal{B}} \subseteq (E^A \times E^{\mathcal{B}})^\omega$ such that $((p_0, a_1, p'_1), (q_0, b_1, q'_1))\ ((p_1, a_2, p'_2), (q_1, a_2, q'_2))\ldots \in (E^A \times E^{\mathcal{B}})^\omega$ is in $L_{\mathcal{A},\mathcal{B}}$ iff either*

- *$p'_i \neq p_i$ for some $i > 0$,*
- *$p'_i = p_i$ for all $i > 0$ and $p_0a_1p_1\ldots \notin \mathsf{AccRun}(\mathcal{A})$, or*
- *$q'_i = q_i$, $a_i = b_i$ for all $i > 0$ and $q_0b_1q_1\ldots \in \mathsf{AccRun}(\mathcal{B})$.*

$L_{\mathcal{A},\mathcal{B}}$ is solvable with a delay function with constant 1 iff $\mathcal{A} \sqsubseteq \mathcal{B}$.

Proof. Suppose Duplicator wins the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Let f be a delay function with constant 1, i.e. $f(i) = 1$ for all $i \in \mathbb{N}$. In the delay simulation game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$, Duplicator plays as follows. Suppose in the beginning of round $i + 1$, Spoiler and Duplicator have formed the words $(p_0, a_1, p'_1)\ \ldots\ (p_{i-1}, a_i, p'_i) \in (E^{\mathcal{A}})^*$ and $(q_0, b_1, q'_1)\ \ldots\ (q_{i-1}, b_i, q'_i) \in (E^{\mathcal{B}})^*$, and Spoiler extends his word with $(p_i, a_{i+1}, p'_{i+1}) \in E^{\mathcal{A}}$. Hence we reach the configuration $(i+1, (p_i, a_{i+1}, p'_{i+1}), \mathsf{D})$. In the case where there is j, $1 \leq j \leq i$, such that $p'_j \neq p_j$, Duplicator simply extends her word by choosing some arbitrary transition $(q, b, q') \in E^{\mathcal{B}}$. Otherwise, we have $p'_j = p_j$ for all $1 \leq j \leq i$ and $r = p_0a_1p_1\ldots p_ia_{i+1}p'_i$ is a valid run in $\mathcal{A}$. In this case, Duplicator considers what she would do in $\mathcal{G}(\mathcal{A}, \mathcal{B})$ if Spoiler has formed the run r and she has formed the run $r' = q_0b_1q_1\ldots q_i$. If Duplicator extends her run to q_{i+1} by reading a_{i+1} then in the game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$, Duplicator continues by extending her word with (q'_i, a_{i+1}, q_{i+1}), i.e. she proceeds to $(i + 2, (q'_i, a_{i+1}, q_{i+1}), \mathsf{S})$.

Now suppose Spoiler and Duplicator have formed infinite words $\rho = (p_0, a_1, p'_1)$ $(p_1, a_2, p'_2)\ldots$ and $\rho' = (q_0, b_1, q'_1)(q_1, b_2, q'_2)\ldots$. There are three cases: either ρ does not correspond to a valid run, i.e. there is $i > 0$ such that $p'_i \neq p_i$, ρ corresponds to a valid run but not accepting, i.e. $p'_i = p_i$ for all $i \in \mathbb{N}$, but $p_0 a_1 p_1 \ldots \notin \mathsf{AccRun}(\mathcal{A})$, or ρ corresponds to a valid and accepting run in $\mathcal{A}$. In the first two cases, by definition of $L_{\mathcal{A},\mathcal{B}}$, Duplicator wins. In the third case, since Duplicator plays according to the wining strategy in $\mathcal{G}(\mathcal{A},\mathcal{B})$, we have $a_i = b_i$ for all $i \in \mathbb{N}$ and $q_0 b_1 q_1 \ldots$ is accepting. Hence $((p_0, a_1, p'_1), (q_0, b_1, q'_1))\ ((p_1, a_2, p'_2), (q_1, b_2, q'_2)) \ldots$ is in $L_{\mathcal{A},\mathcal{B}}$. Duplicator also wins the game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$. We can also show similarly that Duplicator wins $\mathcal{G}(\mathcal{A},\mathcal{B})$ if she wins the game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$. □

Delay simulation can be used to approximate language inclusion. If there exists $d \in \mathbb{N}$ such that the language $L_{\mathcal{A},\mathcal{B}}$ is solvable with a delay function with constant d then language inclusion holds.

Corollary 2.6.29. *For any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ if $L_{\mathcal{A},\mathcal{B}}$ is solvable with some delay function with constant d then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

Proof. For any word $w = a_1 a_2 \ldots \in L(\mathcal{A})$, let $\rho = p_0 a_1 p_1 \ldots$ be an accepting run over w in $\mathcal{A}$. Let f be some delay function with constant d such that $L_{\mathcal{A},\mathcal{B}}$ is solvable with f. We then consider the game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$, assuming that Spoiler forms a word $(p_0, a_1, p_1)(p_1, a_2, p_2) \ldots$ and Duplicator plays according to the winning strategy. Since Duplicator wins, she will form $(q_0, a_1, q_1)(q_1, a_2, q_2) \ldots$ such that $((p_0, a_1, p_1), (q_0, a_1, q_1))$ $((p_1, a_2, p_2), (q_1, a_2, q_2)) \ldots \in L_{\mathcal{A},\mathcal{B}}$. Since $p_0 a_1 p_1 \ldots$ is an accepting run in $\mathcal{A}$, by definition of $L_{\mathcal{A},\mathcal{B}}$, $q_0 a_1 q_1 \ldots$ is an accepting run in $\mathcal{B}$. Hence $w \in L(\mathcal{B})$. □

In the next chapter, we will introduce our notion of buffered simulation and compare it with all the extended simulations that have been mentioned in this chapter. We will further show the advantage and disadvantage of using the extended simulations when approximating language inclusion in comparison to buffered simulation in Chapter 5.

Chapter 3

Buffered Simulation

In this chapter, we give the formal definition of buffered simulation which is the main topic of this work. We will start with a simple case of buffered simulation where only one buffer is involved. In such a case, buffered simulation is a simple extension of standard fair simulation. It extends the game framework of the standard fair simulation such that DUPLICATOR can postpone her move and use the buffer to temporarily store letter that is read by SPOILER, before she executes it in her structure. We will consider some variants of such a simulation game and show how they relate to the extended simulations that we have listed in the previous chapter.

From the case of one buffer, we will consider its natural extension to the case where multiple buffers are involved. DUPLICATOR can use several buffers to store SPOILER's letters in which each letter determines to which buffers it should be stored. We will show various examples of buffered simulation with multiple buffers and also its expressive power.

3.1 Simulation with One Buffer

To illustrate buffered simulation with one buffer, consider a simulation game where a FIFO buffer is available throughout the play. SPOILER plays by moving his pebble one step in each round as in the standard fair simulation, but DUPLICATOR is allowed to skip her turn and use the buffer to store SPOILER's letter. For simplicity, let us assume that the capacity of the buffer is $k \in \mathbb{N}$. This simply means that DUPLICATOR can only use the buffer to store at most k many letters. Intuitively, a play proceeds as follows. In each round, SPOILER moves his pebble one step in $\mathcal{A}$ by reading a letter, suppose a, and then pushes a copy of a to the buffer. DUPLICATOR responds to this by either skipping her turn and doing nothing, or popping some letters from the buffer, suppose $a_1, \ldots, a_n$, and then moving her pebble in $\mathcal{B}$, n steps, by reading $a_1 \ldots a_n$. After DUPLICATOR's turn, the buffer should not contain more than k many letters, otherwise DUPLICATOR loses immediately. The play then proceeds to the next round.

The winning condition in the buffered simulation game is the same as in the standard fair simulation game. DUPLICATOR wins iff SPOILER eventually gets stuck or DUPLICATOR forms an accepting run whenever SPOILER forms one. We can formally define buffered simulation with one buffer as follows.

Definition 3.1.1. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ and $k \in \mathbb{N}$. The *buffered simulation game with one buffer* of capacity k is $\mathcal{G}^k(\mathcal{A}, \mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where SPOILER's and

Duplicator's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times \Sigma^{\leq k} \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{S}\},$$
$$V_0 = Q^{\mathcal{A}} \times \Sigma^{\leq k+1} \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

$$(p, w, q, b, \mathsf{S}) \to (p', wa, q, b, \mathsf{D}) \text{ in } E \quad \text{iff} \quad p \xrightarrow{a} p',$$

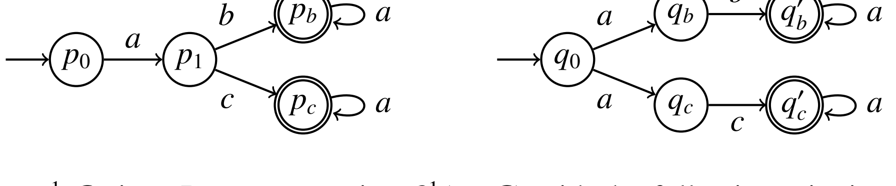
$$(p, w, q, b, \mathsf{D}) \to (p, w', q', b', \mathsf{S}) \text{ in } E \quad \text{iff} \quad \exists u = a_1 \dots a_n \in \Sigma^* \text{ such that } q \xrightarrow{u} q', \quad w = uw', \text{ and } \quad b' = \begin{cases} \top & \text{if } u \neq \epsilon \text{ and } q \xrightarrow[F]{u} q' \\ \bot & \text{otherwise,} \end{cases}$$

the initial configuration is $v_0 = (p_0, \epsilon, q_0, \bot, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$ and an infinite play $v_0 v_1 \dots \in \mathsf{Win}$ iff there exist infinitely many i such that $v_i = (p, w, q, \top, \mathsf{D})$ or there are only finitely many i such that $v_i = (p, w, q, b, \mathsf{S})$ with $p \in F^{\mathcal{A}}$. We write $\mathcal{A} \sqsubseteq^k \mathcal{B}$ if Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$.

In the configuration of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, the first and the third components respectively represent the position of Spoiler's and Duplicator's pebbles, the second one represents the content of the buffer, the fourth one remembers whether Duplicator has moved through an accepting state, and the last one tells us which player has the next turn. It is not necessary to remember whether Spoiler has moved through an accepting state since he only moves one step in each round. We can simply determine this by looking at Spoiler's current state.

Note that we include the requirement where Duplicator should obey the capacity restriction by the definition of a valid configuration. Spoiler's configuration (p, w, q, b, S) is only valid if $|w| \leq k$. Hence Duplicator can only proceed to a configuration where the buffer contains at most k many letters. If such a move is not possible then Duplicator gets stuck and loses the play immediately. However, since we only check the capacity restriction after Duplicator's move, the configuration (p, w, q, b, D) is valid if $|w| \leq k + 1$. This intuitively means that Spoiler can push one more letter to a "full" buffer and it is Duplicator's responsibility to shorten the buffer again. We illustrate this condition in the following example.

Example 3.1.2. Consider again the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 2.5.6. For convenience, we present them again as follows.



We have $\mathcal{A} \sqsubseteq^1 \mathcal{B}$ since Duplicator wins $\mathcal{G}^1(\mathcal{A}, \mathcal{B})$ with the following winning strategy. After Spoiler proceeds from the initial configuration $(p_0, \epsilon, q_0, \bot, \mathsf{S})$ to $(p_1, a, q_0, \bot, \mathsf{D})$ by reading a, Duplicator skips her turn. Hence at the end of the first round, we reach the configuration $(p_1, a, q_0, \bot, \mathsf{S})$. Spoiler then continues to some configuration $(p_x, ax, q_0, \bot, \mathsf{D})$

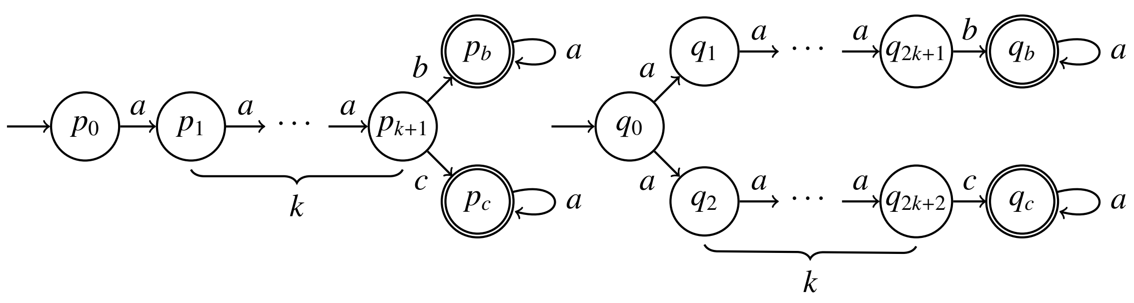


Figure 3.1: NBA $\mathcal{A}_k$, $\mathcal{B}_k$ in which $\mathcal{A}_k \sqsubseteq^{k+1} \mathcal{B}_k$, but $\mathcal{A}_k \not\sqsubseteq^k \mathcal{B}_k$.

by reading some $x \in \{b, c\}$. Duplicator responds to this by reading ax and proceeds to $(p_x, \epsilon, q'_x, \top, \mathsf{S})$, i.e. she pops ab or ac from the buffer by going to the accepting state q'_b or q'_c respectively. From the configuration $(p_x, \epsilon, q'_x, \top, \mathsf{S})$, it is not hard to see that Duplicator can continue accordingly and win the play.

In the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, the capacity constraint has to be satisfied only after Duplicator's turn. Hence it is also possible to consider buffered simulation where the capacity of the buffer is 0. Intuitively, this just means that Duplicator has to pop the letter that is pushed by Spoiler immediately. In fact, buffered simulation with one buffer of capacity 0 is equivalent to the ordinary simulation. We formally show this as follows.

Theorem 3.1.3. $\sqsubseteq = \sqsubseteq^0$.

Proof. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA and suppose Duplicator wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$. In the game $\mathcal{G}^0(\mathcal{A}, \mathcal{B})$, Duplicator plays as follows. In any round $i > 0$, if we are at some configuration (p, a, q, b, D) then we look at what Duplicator would do in the game $\mathcal{G}(\mathcal{A}, \mathcal{B})$ if she is at the configuration (p, a, q, D). If she reads a by going to q', i.e. proceeds to (p, q', S), then in $\mathcal{G}^0(\mathcal{A}, \mathcal{B})$, Duplicator does the same. She proceeds to (p, q', b', S) where $b' = \top$ iff $q' \in F^{\mathcal{B}}$. Duplicator moves her pebble in the same way as in in $\mathcal{G}(\mathcal{A}, \mathcal{B})$.

Let $v_0 v_1 \ldots$ be a play that is obtained in the game $\mathcal{G}^0(\mathcal{A}, \mathcal{B})$. Since Duplicator wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$, either there exist infinitely many i such that $v_i = (p_i, q_i, b_i, \mathsf{D})$ with $q_i \in F^{\mathcal{A}}$, or there are infinitely many i such that $v_i = (p_i, a_i, q_i, b_i, \mathsf{S})$ with $q_i \in F^{\mathcal{B}}$ and hence $b_i = \top$. Thus Duplicator wins the play $v_0 v_1 \ldots$. She wins the game $\mathcal{G}^0(\mathcal{A}, \mathcal{B})$ if she wins $\mathcal{G}(\mathcal{A}, \mathcal{B})$. The other direction can also be shown similarly. □

In the buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, Duplicator has a preview of Spoiler's move. The bigger the buffer, the more preview Duplicator can have. Actually, winning is monotone in the amount of information available about the opponent's future moves. Whenever Duplicator wins the buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, she also wins any game $\mathcal{G}^{k'}(\mathcal{A}, \mathcal{B})$ in which $k' \geq k$.

Theorem 3.1.4. $\sqsubseteq^0 \subsetneq \sqsubseteq^1 \subsetneq \sqsubseteq^2 \ldots$

Proof. For any $k \in \mathbb{N}$, we have $\sqsubseteq^k \subseteq \sqsubseteq^{k+1}$. Duplicator can use the winning strategy in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ for the game $\mathcal{G}^{k+1}(\mathcal{A}, \mathcal{B})$.

For the strictness part, consider the automata in Figure 3.1. For all $k \in \mathbb{N}$, Duplicator wins the game $\mathcal{G}^{k+1}(\mathcal{A}_k, \mathcal{B}_k)$. Intuitively, she skips her turn until Spoiler's pebble reaches p_{k+1}. Hence we eventually reach the configuration $(p_{k+1}, a^{k+1}, q_0, \bot, \mathsf{S})$. Spoiler then will read some $x \in \{b, c\}$ and proceed to the configuration $(p_x, a^{k+1}x, q_0, \bot, \mathsf{D})$. Duplicator responds to this by popping all the letters from the buffer and reaches the corresponding accepting state, i.e. she proceeds to $(p_x, \epsilon, q_x, \top, \mathsf{D})$. From such a configuration, DUPLICATOR can play accordingly and win the play.

DUPLICATOR, however, loses the game $\mathcal{G}^k(\mathcal{A}_k, \mathcal{B}_k)$. In $\mathcal{G}^k(\mathcal{A}_k, \mathcal{B}_k)$, the winning strategy for SPOILER is to initially move his pebble to p_{k+1}. Since the capacity of the buffer is k, in some round $k' \leq k+1$, DUPLICATOR moves her pebble. At the end of round $k+1$, we reach some configuration $(p_{k+1}, w, q_i, \bot, \mathsf{S})$ where $w \in a^*$ and $1 \leq i \leq 2k+2$. If i is odd then SPOILER reads c and proceeds to $(p_c, wc, q_i, \bot, \mathsf{D})$, otherwise he reads b and proceeds to $(p_b, wb, q_i, \bot, \mathsf{D})$. Since from any state q_i, if i is odd, there is no c-transition, and if i is even, there is no b-transition, from such configurations, DUPLICATOR eventually gets stuck and loses the play. □

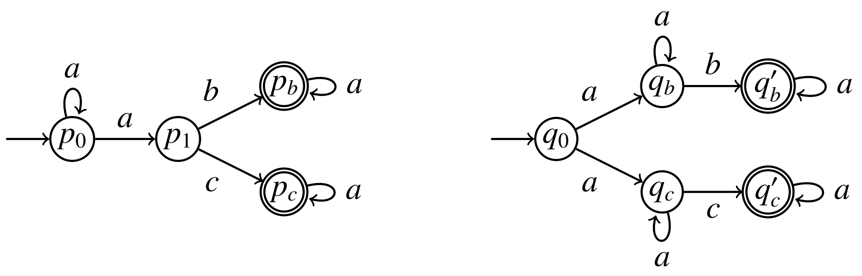
It is also possible to consider buffered simulation in which the capacity of the buffer is unbounded. This simply means that there is no bound on how many letters DUPLICATOR can store to the buffer. She can store as many letters as she wants. Let us denote such a game with $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$. The formal definition of $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ is basically the same as Definition 3.1.1. The only difference is in the definition of SPOILER's and DUPLICATOR's valid configurations. For the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$, we have

$$V_1 = Q^{\mathcal{A}} \times \Sigma^* \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{S}\}, \tag{3.1}$$

$$V_0 = Q^{\mathcal{A}} \times \Sigma^* \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{D}\}. \tag{3.2}$$

We write $\mathcal{A} \sqsubseteq^\omega \mathcal{B}$ if DUPLICATOR wins $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$.

Example 3.1.5. Consider the following two NBA $\mathcal{A}$ and $\mathcal{B}$ that are obtained by slightly modifying the automata from Example 3.1.2.



We have $\mathcal{A} \sqsubseteq^\omega \mathcal{B}$ since DUPLICATOR has a winning strategy in the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ as follows. She skips her turns until SPOILER moves the pebble to the accepting state p_b or p_c, i.e. from any configuration

$$(p, w, q_0, \bot, \mathsf{D}) \tag{3.3}$$

where $w \in \Sigma^*$, if $p \in \{p_0, p_1\}$, DUPLICATOR proceeds to $(p, w, q_0, \bot, \mathsf{S})$.

If SPOILER eventually reaches p_b or p_c, i.e. reaches the configuration (3.3) with $p = p_x$ and $x \in \{b, c\}$, then DUPLICATOR moves her pebble from q_0 to q'_b or q'_c, respectively, by popping all the letters from the buffer. She proceeds to

$$(p_x, \epsilon, q'_x, \top, \mathsf{S}). \tag{3.4}$$

From such a configuration, it is not hard to see that DUPLICATOR can continue accordingly and win the play.

In the case where SPOILER never moves his pebble to p_b or p_c then he does not form an accepting run. We obtain a play $v_0 v'_1 v_1 \dots$ in which for all $i > 0$, $v'_i = (p_0, w, q, b, \mathsf{D})$ and $p_0 \notin F^{\mathcal{A}}$. In such a case, DUPLICATOR wins the play. Hence DUPLICATOR wins the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$.

Buffered simulation with an unbounded buffer includes the one with a bounded buffer. If Duplicator wins $\mathcal{G}^k(\mathcal{A},\mathcal{B})$ for some $k \in \mathbb{N}$ then she also wins $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$ with the same winning strategy. For all $k \in \mathbb{N}$, we have $\sqsubseteq^k \subseteq \sqsubseteq^\omega$. This inclusion however is strict. We formally show this in the following proposition.

Proposition 3.1.6. *For any $k \in \mathbb{N}$, $\sqsubseteq^k \subsetneq \sqsubseteq^\omega$.*

Proof. For the strictness part, consider the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 3.1.5. For any $k \in \mathbb{N}$, Spoiler wins the game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$. The winning strategy is as follows. He loops in p_0 by reading a until Duplicator moves to q_b or q_c. In other words, from any configuration $(p_0, w, q_0, \bot, \mathsf{S})$ where $w \in a^{\leq k}$, Spoiler proceeds to $(p_0, wa, q_0, \bot, \mathsf{D})$. Since the capacity of the buffer is bounded by k, there is a round $k' \leq k+1$ where Duplicator leaves q_0. The play eventually reaches a configuration $(p_0, w', q_x, \bot, \mathsf{S})$ where $w' \in a^{\leq k}$ and $x \in \{b, c\}$, In such a case, Spoiler continues to $p_{\overline{x}}$ where $\overline{x} \in \{b, c\} \setminus \{x\}$ by reading $a\overline{x}$, i.e. he proceeds to $(p_1, w'a, q_x, \bot, \mathsf{D})$ by reading a, and after Duplicator continues to some configuration $(p_1, w'', q_x, \bot, \mathsf{S})$ by skipping her turn or popping some as from the buffer, he proceeds to

$$(p_{\overline{x}}, w''\overline{x}, q_x, \bot, \mathsf{S}) \tag{3.5}$$

by reading $\overline{x}$.

From such a configuration, Spoiler continues by looping in $p_{\overline{x}}$ for the rest of the play. Since from q_x there is no $\overline{x}$-transition, Duplicator eventually will get stuck and lose the play.

Duplicator loses the game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$ for all $k \in \mathbb{N}$. However, from Example 3.1.5, we have seen that she wins $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$. Hence we have the desired property. □

Buffered simulation with an unbounded buffer does not characterise language inclusion. There are cases in which Duplicator loses the buffered simulation game with an unbounded buffer, but language inclusion holds.

Corollary 3.1.7. *There are two NBA $\mathcal{A}$, $\mathcal{B}$ such that $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^\omega \mathcal{B}$.*

Proof. Consider the pair of automata $\mathcal{A}'$, $\mathcal{B}'$ that are obtained from $\mathcal{A}$, $\mathcal{B}$ from Example 3.1.5 by considering all states to be accepting. Hence $L(\mathcal{A}') = L(\mathcal{B}') = a^+ \cdot (a^\omega \cup (b \cup c) \cdot a^\omega)$. However, we have $\mathcal{A}' \not\sqsubseteq^\omega \mathcal{B}'$. The winning strategy for Spoiler in the game $\mathcal{G}^\omega(\mathcal{A}',\mathcal{B}')$ is similar to the one that we have described in the proof of Proposition 3.1.6. Intuitively, Spoiler loops in the initial state of $\mathcal{A}'$ indefinitely until Duplicator leaves the initial state of $\mathcal{B}'$. If Duplicator eventually leaves the initial state then we can show similarly as in the proof of Proposition 3.1.6 that she eventually gets stuck and lose the play. However, if Duplicator never leaves the initial state then she does not form an accepting run, but Spoiler forms one. We obtain a play $v_0 v_1 \ldots$ in which there are only finitely many i such that $v_i = (p, w, q, \top, \mathsf{S})$, but there are infinitely many i such that $v_i = (p, w, q, b, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. Hence Spoiler wins $\mathcal{G}^\omega(\mathcal{A}',\mathcal{B}')$. □

Any buffered simulation game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$ with $k \in \mathbb{N} \cup \{\omega\}$, can be seen as a parity game. Player 1 corresponds to Spoiler and player 0 to Duplicator. It is not hard to see that the winning condition of the buffered simulation game can be expressed as a parity condition.

Theorem 3.1.8. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and $k \in \mathbb{N} \cup \{\omega\}$, we can construct in polynomial time a parity game $\mathcal{G}$ with priorities 0, 1, and 2 such that* Duplicator *wins $\mathcal{G}^k(\mathcal{A},\mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

Proof. Let $G = (V, V_0, V_1, E)$ be the configuration graph of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ and v_0 the initial configuration of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. Consider the parity game $\mathcal{G} = (G, v_0, \Omega)$ where Ω is a priority function that mimics the winning condition of Duplicator in the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, i.e.

$$\Omega(v) = \begin{cases} 2 & \text{if } v = (p, w, q, \top, \mathsf{S}) \\ 1 & \text{if } v = (p, w, q, b, \mathsf{D}) \text{ and } p \in F^{\mathcal{A}} \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

Suppose Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. In the parity game $\mathcal{G}$, player 0 plays according to the winning strategy for Duplicator in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. Let $\pi = v_0 v_1 \ldots$ be a play that is obtained in $\mathcal{G}$. Since Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, either there exist infinitely many i such that $v_i = (p, w, q, \top, \mathsf{S})$ or there are only finitely many i such that $v_i = (p, w, q, \top, \mathsf{D})$. In the first case, the highest priority that is seen infinitely often is 2 and in the second one, it is 0. Hence player 0 wins $\mathcal{G}$. We can also show the other direction similarly. □

This shows that the buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, $k \in \mathbb{N} \cup \{\omega\}$, can be seen as a parity game by considering a priority function as in (3.6). Recall that in a parity game, the winner always has a memoryless strategy. Thus the winner of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, $k \in \mathbb{N} \cup \{\omega\}$, also has a memoryless winning strategy.

There is also a special kind of winning strategy for Duplicator if she ever wins a buffered simulation game. If Duplicator wins the buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N} \cup \{\omega\}$, she also has a winning strategy in which she only pops at most one letter in each round. We can convert any winning strategy in the buffered simulation game to such a special one. Intuitively, if the strategy tells Duplicator to move her pebble by popping several letters from the buffer then Duplicator remembers the path that she should have taken and moves the pebble in the next rounds by popping the letters one by one.

To formally show this, let $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ be a variant of buffered simulation game where in every round, Duplicator moves her pebble at most only one step. The formal definition of $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ is basically the same as the one in Definition 3.1.1, but with $|u| \leq 1$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{One}} \mathcal{B}$ if Duplicator wins $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$.

Theorem 3.1.9. $\sqsubseteq^k_{\mathsf{One}} = \sqsubseteq^k$.

Proof. The left-to-right direction is trivial since if Duplicator wins $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ then Duplicator also wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ by considering the same winning strategy.

For the other direction, suppose Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. Let σ be a memoryless winning strategy for Duplicator in the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. In $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$, from the initial configuration, Duplicator plays according to σ until at one point the strategy tells Duplicator to pop $n > 1$ many letters, i.e. to proceed from some configuration $(p, a_1 \ldots a_m, q, b, \mathsf{D})$ to

$$(p, a_{n+1} \ldots a_m, q_n, b', \mathsf{S}), \tag{3.7}$$

by taking some path $q a_1 q_1 \ldots a_n q_n$. In the game $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$, Duplicator pops the letters $a_1, \ldots, a_n$ one by one. Note that if during such moves, Spoiler moves his pebble along the path $r = p a_{m+1} p_1 \ldots a_{m+n} p_n$ then from the configuration $(p, a_1 \ldots a_m, q, b, \mathsf{D})$, the play proceeds successively to the configurations:

$$(p, a_2 \ldots a_m, q_1, b_1, \mathsf{S}), (p_1, a_2 \ldots a_{m+1}, q_1, b_1, \mathsf{D}), \ldots, (p_n, a_{n+1} \ldots a_{m+n}, q_k, b_k, \mathsf{D}),$$

where $b_i = \top$ iff $q_i \in F^B$ for all $i \in \{1, \dots, k\}$. DUPLICATOR then looks again what she would do in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ if from the configuration (3.7), SPOILER continues by moving the pebble along the path r. She then repeats this procedure indefinitely.

Let $\pi = v_0 v_1 \dots$ be the play that is obtained in the game $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$. Since DUPLICATOR forms the same run that she would have formed in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ and she wins the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, there exist infinitely many i such that $v_i = (p, w, q, \top, \mathsf{D})$ or there are only finitely many i such that $v_i = (p, w, q, b, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. Hence DUPLICATOR also wins the play π. She wins the game $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$. □

In the case where we consider a game with a bounded buffer, there is even a more restricted winning strategy for DUPLICATOR if she ever wins the buffered simulation game. If DUPLICATOR wins the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ in which $k \in \mathbb{N}$, there is a winning strategy where DUPLICATOR skips her turn for the first k rounds and pops only one letter in each round for the rest of the play. Note that if DUPLICATOR plays according to such a strategy then after the k-th round, the buffer is always full, i.e. it always contains k many letters at the end of the round. In each round $i > k$, DUPLICATOR always forms a run that is k steps behind SPOILER's run.

We can convert any DUPLICATOR's winning strategy in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, where $k \in \mathbb{N}$, to such a special one. Intuitively, if the strategy tells DUPLICATOR to move her pebble in some round $i \leq k$, DUPLICATOR remembers the path that she should have taken and only moves the pebble after round k, step by step. To show this formally, let $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$, $k \in \mathbb{N}$, be a restricted variant of buffered simulation where DUPLICATOR's move is restricted such that she has to skip her turn in the first k rounds and move the pebble one step in each round for the rest of the play. The formal definition of $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$ is the same as the one in Definition 3.1.1, but with $|u| = 0$ if $|w| \leq k$ and $|u| = 1$ if $|w| = k + 1$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{Full}} \mathcal{B}$ if DUPLICATOR wins $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$. In the following, we show that the game $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$ and the original buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ are indeed equivalent.

Theorem 3.1.10. $\sqsubseteq^k_{\mathsf{Full}} = \sqsubseteq^k$.

Proof. The left-to-right direction is trivial. For the right-to-left direction, by Theorem 3.1.9, it suffices to show that $\sqsubseteq^k_{\mathsf{One}} \subseteq \sqsubseteq^k_{\mathsf{Full}}$. Now suppose DUPLICATOR wins $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$. The winning strategy for DUPLICATOR in $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$ is as follows. Initially, she skips her turns for the first k rounds. Suppose in the first k rounds, SPOILER moves his pebble along the run $p_0 a_1 p_1 \dots p_k$ and in round $k + 1$, he goes to p_{k+1} by reading a_{k+1}. Hence in round $k + 1$, we are in the configuration

$$(p_{k+1}, a_1 \dots a_{k+1}, q_0, \bot, \mathsf{D}). \tag{3.8}$$

DUPLICATOR then looks at what she would do in $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ if from the initial configuration, SPOILER moves the pebble along the path $r = p_0 a_1 p_1 \dots p_{k+1}$. If DUPLICATOR responds to this by moving the pebble along $q_0 a_1 q_1 \dots q_n$ and proceeding to the configuration

$$(p_{k+1}, a_{n+1} \dots a_{k+1}, q_n, b_n, \mathsf{D}) \tag{3.9}$$

then in the game $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$, DUPLICATOR pops the letters $a_1, \dots, a_n$ one by one. If during such moves, SPOILER moves his pebble along the path $r' = p_{k+1} a_{k+2} p_{k+2} \dots a_{k+n+1} p_{k+n+1}$ then from the configuration (3.8), the play proceeds successively to the configurations:

$$(p_{k+1}, a_2 \dots a_{k+1}, q_1, b_1, \mathsf{S}), (p_{k+1}, a_2 \dots a_{k+2}, q_1, b_1, \mathsf{D}), \dots, (p_{k+n+1}, a_{n+1} \dots a_{k+n+1}, q_n, b_n, \mathsf{D}),$$

where $b_i = \top$ iff $q_i \in F^B$ for all $i \in \{1,\dots,k\}$. Duplicator again looks at what she would do in $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A},\mathcal{B})$ if from configuration (3.9), Spoiler moves along r'. She then repeats the same procedure for the rest of the play.

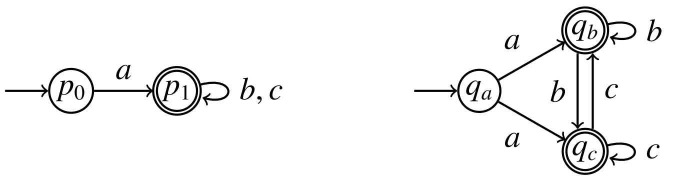
Since Duplicator forms the same run that she would have formed in $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A},\mathcal{B})$ and she wins $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A},\mathcal{B})$, in the play $v_0v_1\dots$ that is formed in $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A},\mathcal{B})$, there are infinitely many i with $v_i = (p,w,q,\top,\mathsf{S})$ or finitely many i such that $v_i = (p,w,q,b,\mathsf{D})$ with $p \in F^{\mathcal{A}}$. Duplicator also wins the game $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A},\mathcal{B})$. □

Hence in the buffered simulation game with one buffer of capacity $k \in \mathbb{N}$, if Duplicator wins the game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$, we can assume that there is a winning strategy that initially lets the buffer fill up with k many letters and then moves alternatingly with Spoiler by popping one letter in each round for the rest of the play.

3.2 The Flushing Variant

Another interesting variant of buffered simulation game is the one where Duplicator is required to pop all the letters from the buffer each time she moves the pebble. Hence afer Duplicator moves, the buffer is always empty. Let us denote such a game $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ and call it the flushing variant. Moreover, let us also call the move where Duplicator pops all the letters from the buffer *flushing*. The formal definition of $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ only differs from Definition 3.1.1 in the length of the word u that is chosen by Duplicator. We either have $|u| = 0$ or $|u| = |w|$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$ if Duplicator wins the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$.

Example 3.2.1. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ that we have presented in Figure 1.2. For convenience, we present the automata again as follows.



Duplicator loses the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ for any $k \in \mathbb{N}$. Spoiler wins with the following winning strategy. He initially goes to p_1 and loops there by reading b until at one point Duplicator flushes the buffer and leaves the initial state. Formally, from the initial configuration $(p_0,\epsilon,q_a,\bot,\mathsf{S})$, Spoiler proceeds to $(p_1,a,q_a,\bot,\mathsf{D})$ and from any configuration $(p_1,w,q_a,\bot,\mathsf{S})$ where $w \neq \epsilon$, he continues to $(p_1,wb,q_a,\bot,\mathsf{D})$.

Since the capacity of the buffer is bounded by k, there is some round $k' \leq k+1$ where Duplicator flushes the buffer. Hence at round $k'+1$, we reach some configuration $(p_1,\epsilon,$ $q_x,\top,\mathsf{S})$ where $x \in \{b,c\}$. Spoiler then continues by reading $\overline{x}$ where $\overline{x} \in \{b,c\} \setminus \{x\}$. He proceeds to

$$(p_1,\overline{x},q_x,\top,\mathsf{D}). \tag{3.10}$$

Spoiler then reads b by looping in p_1 for the rest of the play. Duplicator eventually gets stuck because there is no $\overline{x}$-transition from q_x. Thus Spoiler wins $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$.

Similarly to the general case, we can also consider the flushing variant for the case where we consider an unbounded buffer. For any NBA $\mathcal{A}$, $\mathcal{B}$, let us denote such a game with $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$. The definition of $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ is the same as $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ where $k \in \mathbb{N}$.

The only difference is in the definition of Spoiler's and Duplicator's valid configurations. They are respectively defined as in (3.1) and (3.2). We write $\mathcal{A} \sqsubseteq^{\omega}_{\mathsf{Flush}} \mathcal{B}$ if Duplicator wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$.

Example 3.2.2. Consider again the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 3.2.1. Duplicator also loses the flushing variant $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$. The winning strategy for Spoiler is the same as in Example 3.2.1. He reads $abbb\ldots$ indefinitely until Duplicator flushes the buffer by going to some state q_x where $x \in \{b, c\}$. Spoiler then reads $\overline{x} \in \{b, c\} \setminus \{x\}$ to make Duplicator get stuck.

In this case, Duplicator additionally might skip her turn forever and never flushes the buffer. However Spoiler still wins in such a case since he forms an accepting run and Duplicator does not. We obtain a play $v_0 v'_1 v_1 \ldots$ where for all $i > 0$, we have $v_i = (p_1, w, q_a, \bot, \mathsf{S})$ and $v'_i = (p_1, w, q_a, \bot, \mathsf{D})$ where $w \in ab^*$. Since $p_1 \in F^{\mathcal{A}}$, Spoiler wins the play. He wins the game $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$.

Unlike the variants $\mathcal{G}^{k}_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{k}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$, the flushing variant $\mathcal{G}^{k}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is not equivalent to the original buffered simulation game. We do not have the inclusion $\sqsubseteq^{k} \subseteq \sqsubseteq^{k}_{\mathsf{Flush}}$ because there are cases in which Duplicator wins the general buffered simulation game, but loses the corresponding flushing variant.

Theorem 3.2.3. *For any $k \in \mathbb{N}^+ \cup \{\omega\}$, $\sqsubseteq^{k}_{\mathsf{Flush}} \subsetneq \sqsubseteq^{k}$.*

Proof. The inclusion part is trivial since any winning strategy in $\mathcal{G}^{k}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is also a winning strategy in $\mathcal{G}^{k}(\mathcal{A}, \mathcal{B})$. For the strictness part, consider the two NBA $\mathcal{A}$, $\mathcal{B}$ in Example 3.2.1. We have seen that Duplicator loses the flushing variant $\mathcal{G}^{k}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ for any $k \in \mathbb{N} \cup \{\omega\}$. However, she wins the general buffered simulation game $\mathcal{G}^{k}(\mathcal{A}, \mathcal{B})$ with $k = 1$. Initially, Spoiler will read a in the first round. He will proceed from the initial configuration $(p_0, \epsilon, q_a, \bot, \mathsf{S})$ to $(p_1, a, q_a, \bot, \mathsf{D})$. Duplicator then skips her turn. She proceeds to the configuration $(p_1, a, q_a, \bot, \mathsf{S})$. Now from any configuration

$$(p_1, x, q_x, b_x, \mathsf{S}) \tag{3.11}$$

where $x \in \{a, b, c\}$ and $b_x \in \{\top, \bot\}$, Spoiler will read b or c and continue to the configuration $(p_1, xb, q_x, b_x, \mathsf{D})$ or $(p_1, xc, q_x, b_x, \mathsf{D})$, respectively. In the first case, Duplicator pops x from the buffer and proceeds to $(p_1, b, q_b, \top, \mathsf{S})$ and in the second one to $(p_1, c, q_c, \top, \mathsf{S})$. Hence we are back again to the configuration of the form (3.11). Duplicator repeats this procedure indefinitely. In other words, Duplicator skips her turn in the first round and then moves the pebble one step for the rest of the play according to the letter that is pushed by Spoiler to the buffer. If Spoiler pushes b then Duplicator pops one letter from the buffer and moves from her current state to q_b. However, if Spoiler pushes c then she moves to q_c. Thus in each round, Duplicator's has a preview of Spoiler's move one step. This enables her to move the pebble accordingly. She will never get stuck. Duplicator will form an accepting run, i.e. we obtain a play $v_0 v'_1 v_1 v'_2 \ldots$ where for all $i > 1$, we have $v_i = (p, w, q, \top, \mathsf{S})$. Hence Duplicator wins the play. He wins the game $\mathcal{G}^{1}(\mathcal{A}, \mathcal{B})$. By Theorem 3.1.4 and Proposition 3.1.6, she also wins the game $\mathcal{G}^{k}(\mathcal{A}, \mathcal{B})$ for all $k \in \mathbb{N}^+ \cup \{\omega\}$. □

The flushing variant also admits a hierarchy with respect to the capacity of the buffer as in the general case. It is not hard to see that the inclusions $\sqsubseteq^{k}_{\mathsf{Flush}} \subseteq \sqsubseteq^{k+1}_{\mathsf{Flush}}$ and $\sqsubseteq^{k}_{\mathsf{Flush}} \subseteq \sqsubseteq^{\omega}_{\mathsf{Flush}}$ hold for all $k \in \mathbb{N}$. The inclusions are indeed strict since the examples that are used for the general case in the proof of Theorem 3.1.4 and Theorem 3.1.6 also work for the flushing variant. Thus we have the following theorem.

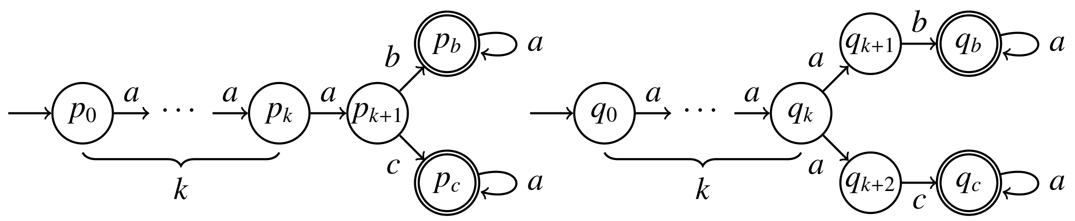


Figure 3.2: NBA $\mathcal{A}_k$, $\mathcal{B}_k$ in which $\mathcal{A} \not\sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$, but $\mathcal{A} \sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$.

Theorem 3.2.4. *For all $k \in \mathbb{N}$,*

- $\sqsubseteq^k_{\mathsf{Flush}} \subsetneq \sqsubseteq^{k+1}_{\mathsf{Flush}}$.
- $\sqsubseteq^k_{\mathsf{Flush}} \subsetneq \sqsubseteq^{\omega}_{\mathsf{Flush}}$.

The Full-Flushing Variant

Beside the flushing variant, we can also consider a more restricted variant where Duplicator is only allowed to flush the buffer after the buffer is full. This variant of course only makes sense if the capacity of the buffer is bounded. Otherwise, the buffer will never be full. Let us call such a variant *full-flushing variant* and denote it with $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$, $k \in \mathbb{N}$. The full-flushing variant proceeds exactly like the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, but in every round, Duplicator skips her turn if Spoiler does not push a letter into a full buffer, otherwise she flushes the buffer. The formal definition of $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$ only differs from Definition 3.1.1 in the length of the word u that is chosen by Duplicator. We have $|u| = 0$ if $|w| \leq k$ and $|u| = |w|$ if $|w| = k + 1$. We write $\mathcal{A} \sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$ if Duplicator wins the full-flushing variant $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$.

The full-flushing variant is weaker than the ordinary flushing variant. There are cases where Duplicator loses the full-flushing variant but wins the ordinary flushing variant. We will show this in general, i.e. for any $k \in \mathbb{N}$, there is a pair of automata $\mathcal{A}$, $\mathcal{B}$ such that Duplicator loses the full-flushing variant $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$, but wins the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$.

Theorem 3.2.5. *For any $k > 0$, $\sqsubseteq^k_{\mathsf{FFlush}} \subsetneq \sqsubseteq^k_{\mathsf{Flush}}$.*

Proof. The inclusion is trivial. For the strictness part, let $k > 0$ and consider two NBA $\mathcal{A}_k$, $\mathcal{B}_k$ as given in Figure 3.2. We have $\mathcal{A}_k \not\sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}_k$. Spoiler has a winning strategy in the game $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}_k, \mathcal{B}_k)$. He first reads a^{k+1}. Since Duplicator is only allowed to flush the buffer when the buffer is filled with $k + 1$ many letters, the play eventually proceeds to the configuration $(p_{k+1}, a^{k+1}, q_0, \bot, \mathsf{D})$. From this configuration, Duplicator has no choice except to flush the buffer. She reads a^{k+1} and proceeds to $(p_{k+1}, \epsilon, q_{k+1}, \bot, \mathsf{S})$ or $(p_{k+1}, \epsilon, q_{k+2}, \bot, \mathsf{S})$. In the first case, Spoiler continues to $(p_c, c, q_{k+1}, \bot, \mathsf{D})$ by reading c and in the second one to $(p_b, b, q_{k+2}, \bot, \mathsf{D})$ by reading b. From such configurations, Duplicator eventually will get stuck since the state q_{k+1} only has a b-transition and the state q_{k+2}, a c-transition. Hence Duplicator loses the game $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}_k, \mathcal{B}_k)$.

Duplicator, however, wins the game $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}_k, \mathcal{B}_k)$. She wins with the following strategy. After Spoiler proceeds from the initial configuration $(p_0, \epsilon, q_0, \bot, \mathsf{S})$ to $(p_1, a, q_0, \bot, \mathsf{D})$ by reading a, Duplicator immediately pops a from the buffer. She proceeds to $(p_1, \epsilon, q_1, \bot, \mathsf{S})$. Duplicator then skips her turn until Spoiler reaches either p_b or p_c. We eventually reach the configuration $(p_b, a^k b, p_1, \bot, \mathsf{D})$ or $(p_c, a^k c, p_1, \bot, \mathsf{D})$. In this case, Du-

Duplicator flushes the buffer and proceeds to $(p_b, \epsilon, q_b, \top, \mathsf{S})$ or $(p_c, \epsilon, q_c, \top, \mathsf{S})$ respectively. From this configuration, Duplicator can play accordingly and win the play. □

Unlike the flushing variant and the general case of buffered simulation, the full-flushing variant does not admit a hierarchy as the capacity of the buffer grows. If Duplicator wins the full-flushing variant $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$ then it is not necessary that Duplicator also wins $\mathcal{G}^{k'}_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$ for any $k' \geq k$.

Theorem 3.2.6. *For any $k > 0$, $\sqsubseteq^k_{\mathsf{FFlush}} \not\subseteq \sqsubseteq^{k+1}_{\mathsf{FFlush}}$.*

Proof. Consider again the NBA given in Figure 3.2. For any $k > 0$, we have $\mathcal{A}_{k+1} \sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}_{k+1}$. Duplicator wins the game $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}_{k+1}, \mathcal{B}_{k+1})$ with the following winning strategy. She first waits until the buffer is filled with $k+1$ many letters, i.e. until the play proceeds to $(p_{k+1}, a^{k+1}, q_0, \bot, \mathsf{D})$. From this configuration, Duplicator flushes the buffer and proceeds to the configuration $(p_{k+1}, \epsilon, q_{k+1}, \bot, \mathsf{S})$. Duplicator then waits again until the buffer is filled with $k + 1$ many letters, i.e. until the play proceeds to $(p_b, aba^{k-1}, q_{k+1}, \bot, \mathsf{D})$ or $(p_c, aca^{k-1}, q_{k+1}, \bot, \mathsf{D})$. From such configurations, Duplicator again flushes the buffer and proceeds to the configuration $(p_b, \epsilon, q_b, \top, \mathsf{S})$ or $(p_c, \epsilon, q_c, \top, \mathsf{S})$, respectively. From this configuration, Duplicator can continue accordingly and win the play. Duplicator, however, loses the game $\mathcal{G}^{k+1}_{\mathsf{FFlush}}(\mathcal{A}_{k+1}, \mathcal{B}_{k+1})$ as we have seen in the proof of Theorem 3.2.5. □

The flushing and the full-flushing variants of buffered simulation with one buffer are closely related to the static and dynamic multi-letter simulations. We will investigate their relations, together with the relation of buffered simulation with one buffer to other extended simulations listed in Chapter 2 in the following section.

3.3 Relation to Other Simulations

3.3.1 Static Multi-Letter Simulation

Let us start with the static multi-letter simulation. Recall that in the static k-letter simulation game, at each round, Spoiler moves his pebble k steps and Duplicator follows this by moving her pebble k steps. This is basically equivalent to the the full-flushing variant of buffered simulation where Duplicator has to skip her turn until Spoiler pushes k many letters to the buffer and then flushes the buffer. The static k-letter simulation is indeed equivalent to the full-flushing variant of buffered simulation where the capacity of the buffer is $k - 1$. Note that we have $k - 1$ instead of k since in the game $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$, Spoiler can still push one more letter to a full buffer before Duplicator flushes the buffer.

Theorem 3.3.1. *For any $k > 0$, $\sqsubseteq^k_{\mathsf{Stat}} = \sqsubseteq^{k-1}_{\mathsf{FFlush}}$.*

Proof. Suppose Duplicator wins $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$. The winning strategy for Duplicator in $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$ is as follows. From some configuration $(p, \epsilon, q, b, \mathsf{S})$, if the play eventually proceeds to the configuration $(p_k, a_1 \ldots a_k, q, \bot, \mathsf{D})$ because Spoiler moves his pebble along the run $r = pa_1p_1 \ldots p_k$ then Duplicator looks at what she would do in the game $\mathcal{G}^k_{\mathsf{Stat}}(\mathcal{A}, \mathcal{B})$ if she is in some configuration $(p_k, a_1 \ldots a_k, q, b, \mathsf{D})$ where $b = \top$ iff at least one of $p_1, \ldots, p_k$ is a final state. If Duplicator proceeds to the configuration $(p_k, q', b', \mathsf{S})$ then in $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$, she proceeds to $(p_k, \epsilon, q', b', \mathsf{S})$.

Now let $v_0v_1\dots$ be the play that is obtained in $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A},\mathcal{B})$. Since Duplicator wins $\mathcal{G}^{k}_{\mathsf{Stat}}(\mathcal{A},\mathcal{B})$, either there exist infinitely many i such that $v_i = (p,\epsilon,q,\top,\mathsf{S})$ or there are only finitely many i such that $v_i = (p,w,q,b,\mathsf{D})$ with $p_i \in F^{\mathcal{A}}$. Hence Duplicator also wins the play $v_0v_1\dots$. She wins the game $\mathcal{G}^{k}_{\mathsf{FFlush}}(\mathcal{A},\mathcal{B})$. The other direction can also be shown similarly. □

3.3.2 Dynamic Multi-Letter Simulation

For the dynamic k-letter simulation, we will show similarly that it is included in the flushing variant of buffered simulation where the buffer is of capacity $k-1$. Recall that in the dynamic k-letter simulation game, at each round, Duplicator can choose how long Spoiler should move his pebble. If Duplicator chooses $\ell \le k$ then Spoiler moves his pebble ℓ steps and Duplicator responds to this by moving her pebble also ℓ steps. Any winning strategy in the dynamic k-letter simulation game can be translated to the one for the flushing variant of buffered simulation of capacity $k-1$. Intuitively, if in the dynamic k-letter simulation game Duplicator chooses ℓ then in the flushing variant of buffered simulation, Duplicator skips her turn and lets Spoiler push ℓ many letters to the buffer before Duplicator flushes the buffer. Hence if Duplicator wins the dynamic k-letter game $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$, she also wins the flushing variant $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A},\mathcal{B})$.

Theorem 3.3.2. *For any $k>0$, $\sqsubseteq^{k}_{\mathsf{Dyn}} \subseteq \sqsubseteq^{k-1}_{\mathsf{Flush}}$.*

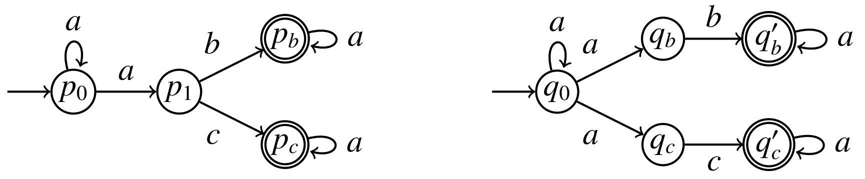
Proof. Suppose Duplicator wins the game $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$. The winning strategy for Duplicator in the game $\mathcal{G}^{k-1}_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ is as follows. If in $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$, Duplicator proceeds from the initial configuration $(p_0,\epsilon,q_0,b,\mathsf{D})$ to $(p_0,q_0,\ell,\bot,\mathsf{S})$ then in the game $\mathcal{G}^{k-1}_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ Duplicator initially skips her turn until Spoiler pushes ℓ many letters to the buffer by moving his pebble along some path $r = pa_1p_1\dots p_\ell$. Hence we reach some configuration $(p_\ell,a_1\dots a_\ell,\,q_0,\bot,\mathsf{D})$. Duplicator then looks at what she would do in $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$ if she is in the configuration $(p_\ell,a_1\dots a_\ell,\,q_0,b,\mathsf{D})$ where $b=\top$ iff at least one of $p_1,\dots,p_\ell$ is final. If Duplicator proceeds to $(p_\ell,q',\ell',b',\mathsf{S})$ then in the game $\mathcal{G}^{k-1}_{\mathsf{FFlush}}(\mathcal{A},\mathcal{B})$, she proceeds to $(p_k,\epsilon,q',b',\mathsf{S})$, waits until Spoiler pushes ℓ' many letters to the buffer, and then repeats the same procedure.

Now let $v_0v_1\dots$ be the play that is obtained in $\mathcal{G}^{k-1}_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$. Since Duplicator wins $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$, either there exist infinitely many i such that $v_i = (p,\epsilon,q,\top,\mathsf{S})$ or there are only finitely many i such that $v_i = (p,w,q,b,\mathsf{D})$ with $p_i \in F^{\mathcal{A}}$. Hence Duplicator also wins the play $v_0v_1\dots$. She wins the game $\mathcal{G}^{k}_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$. □

Note that unlike the static k-letter simulation, if Duplicator wins the flushing variant with buffer of capacity $k-1$, it is not the case that she also wins the dynamic k-letter simulation game. There is even a pair of automata $\mathcal{A}$, $\mathcal{B}$ where Duplicator wins the game $\mathcal{G}^{1}_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$, but loses the game $\mathcal{G}^{k}_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$ for all $k>0$.

Lemma 3.3.3. *There is a pair of NBA $\mathcal{A}$, $\mathcal{B}$ such that $\mathcal{A} \not\sqsubseteq^{k}_{\mathsf{Dyn}} \mathcal{B}$ for all $k>1$, but $\mathcal{A} \sqsubseteq^{1}_{\mathsf{Flush}} \mathcal{B}$.*

Proof. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ that are slightly modified from the ones in Example 3.1.2.



Duplicator wins $\mathcal{G}^1_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$. She simply flushes the buffer whenever Spoiler moves his pebble to p_0 and skips her turn if Spoiler moves to p_1, i.e. from any configuration $(p,a,q_0,\bot,\mathsf{D})$ if $p = p_0$ then she proceeds to $(p_0,\epsilon,q_0,\bot,\mathsf{S})$ and if $p = p_1$ then she proceeds to

$$(p_1,a,q_0,\bot,\mathsf{S}). \tag{3.12}$$

Spoiler eventually will move his pebble to p_1 since otherwise he will lose for not producing an accepting run. Hence in some round, we will reach the configuration (3.12). From this configuration, Spoiler will proceed to some configuration $(p_x,ax,q_0,\bot,\ \mathsf{D})$ where $x \in \{b,c\}$ by reading x. Duplicator then flushes the buffer and continues to $(p_x,\epsilon,q'_x,\top,\mathsf{S})$. From this configuration, it is not hard to see that Duplicator can continue accordingly and win the play.

Duplicator however loses the game $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$ for any $k > 1$. The winning strategy for Spoiler is as follows. In the first round, if Duplicator chooses some $\ell \leq k$, i.e. proceeds from the initial configuration to $(p_0,q_0,\ell,\bot,\mathsf{S})$, then Spoiler loops $\ell-1$ many times in p_0 by reading $a^{\ell-1}$ and then goes to p_1 by reading a. Hence we reach the configuration $(p_1,a^\ell,q_0,\bot,\mathsf{D})$. Duplicator will either reads a^ℓ by looping in q_0, ℓ many times, or looping in q_0, $\ell-1$ many times, and then going to some q_x, $x \in \{b,c\}$ by reading a. In other words, she will either proceed to some configuration

$$(p_1,q_0,\ell',\bot,\mathsf{S}) \text{ or} \tag{3.13}$$
$$(p_1,q_x,\ell',\bot,\mathsf{S}) \tag{3.14}$$

where $\ell' \leq k$. In the first case, Spoiler reads $ba^{\ell'-1}$ by going to p_b. In the second case, he reads $\overline{x}a^{\ell'-1}$ where $\overline{x} \in \{b,c\} \setminus \{x\}$ by going to $p_{\overline{x}}$. Hence the play either proceeds to $(p_b,ba^{\ell'-1},q_0,\top,\mathsf{D})$ from (3.13) or to $(p_{\overline{x}},\overline{x}a^{\ell'-1},q_x,\top,\mathsf{D})$ from (3.14). These configurations however do not have a valid successor since there is no b-transition from q_0 and no $\overline{x}$-transition from q_x. Hence Duplicator loses the game $\mathcal{G}^k_{\mathsf{Dyn}}(\mathcal{A},\mathcal{B})$. □

This nonetheless shows that the inclusion in Theorem 3.3.2 is indeed strict for $k > 1$.

Corollary 3.3.4. *For any $k > 1$, $\sqsubseteq^k_{\mathsf{Dyn}} \subsetneq \sqsubseteq^{k-1}_{\mathsf{Flush}}$.*

Proof. Let $k > 1$ be some number. The inclusion $\sqsubseteq^k_{\mathsf{Dyn}} \subseteq \sqsubseteq^{k-1}_{\mathsf{Flush}}$ holds because of Theorem 3.3.2. Consider the NBA $\mathcal{A}$, $\mathcal{B}$ as in Lemma 3.3.3. We have seen that $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Dyn}} \mathcal{B}$. However, since $\mathcal{A} \sqsubseteq^1_{\mathsf{Flush}} \mathcal{B}$, by Theorem 3.2.4, $\mathcal{A} \sqsubseteq^{k-1}_{\mathsf{Flush}} \mathcal{B}$. Hence $\sqsubseteq^k_{\mathsf{Dyn}} \subsetneq \sqsubseteq^{k-1}_{\mathsf{Flush}}$. □

3.3.3 Multi-Pebble Simulation

Unlike the static and dynamic multi-letter simulations, it is not obvious how buffered simulation relates to multi-pebble simulation. In multi-pebble simulation, Duplicator has to move immediately after Spoiler reads a letter, but she can use multiple pebbles to mimic Spoiler's run. She can drop or duplicate her pebbles before moving them to some corresponding successors. This is in contrast to the buffered simulation game where Duplicator only uses one pebble, but she can skip her turn.

It turns out that the winning strategy in the buffered simulation game can be translated to the one of multi-pebble simulation. Intuitively, the move where Duplicator skips her turn after Spoiler reads a can be mimicked by the move where Duplicator duplicates and moves her pebbles to all of the corresponding a-successors. By doing so, she keeps all the possibilities of extending her run by reading a. Recall that if the capacity of the buffer is k then Duplicator can skip her turn until Spoiler pushes $k+1$ many letters to the buffer. In multi-pebble simulation, we can mimic such a move if Duplicator uses k' many pebbles where k' is the maximum number of states that can be reached by reading a word of length at most $k+1$. By having k' many pebbles, Duplicator can keep duplicating and moving her pebbles to all of the corresponding successors for k consecutive rounds.

Moreover, the move where Duplicator pops ℓ many letters from the buffer and goes to some state q can be mimicked by considering the state q that has been visited by some of her pebbles after ℓ steps. Duplicator simply drops all the pebbles that do not visit q after ℓ many steps. She keeps the ones that were at q, together with other pebbles duplicated from them. By doing so, Duplicator intuitively commits to a certain way of extending her run ℓ steps, namely to the state q, and only considers possible extensions from q.

If Duplicator forms an accepting run ρ in the buffered simulation game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$ then in the corresponding multi-pebble simulation $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A},\mathcal{B})$, Duplicator only keeps pebbles that form ρ. Hence if ρ is accepting then all Duplicator's pebbles that survive infinitely many rounds also see an accepting state infinitely often. Duplicator wins the game $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A},\mathcal{B})$ if she wins $\mathcal{G}^k(\mathcal{A},\mathcal{B})$. This translation also holds for the case of unbounded buffer. In the case where the capacity of the buffer is $k=\omega$ then k' is simply the number of states that are reachable by reading any finite word in $\mathcal{B}$.

Theorem 3.3.5. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over Σ. For any $q \in Q^{\mathcal{B}}$ and $i \in \mathbb{N} \cup \{\omega\}$, let*

$$\mathsf{OutDeg}_i(q) = \begin{cases} |\{q' \mid q \xrightarrow{w} q',\ w \in \Sigma^i\}| & \text{if } i \in \mathbb{N},\\ |\{q' \mid q \xrightarrow{w} q',\ w \in \Sigma^*\}| & \text{if } i = \omega, \end{cases}$$

and

$$k' = \begin{cases} \max_{i\in\{1,\dots,k+1\},\, q\in Q^{\mathcal{B}}}\{\mathsf{OutDeg}_i(q)\} & \text{if } k \in \mathbb{N},\\ \max_{q\in Q^{\mathcal{B}}}\{\mathsf{OutDeg}_\omega(q)\} & \text{if } k = \omega. \end{cases}$$

If $\mathcal{A} \sqsubseteq^k \mathcal{B}$ then $\mathcal{A} \sqsubseteq^{k'}_{\mathsf{Peb}} \mathcal{B}$.

Proof. Suppose Duplicator wins the game $\mathcal{G}^k(\mathcal{A},\mathcal{B})$. By Theorem 3.1.9, Duplicator also wins $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A},\mathcal{B})$. In the game $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A},\mathcal{B})$, from the initial configuration $(p_0, S_0, \emptyset, \mathsf{S})$ where $S_0 = \{q_0\}$, if Spoiler moves along the path $r = p_0a_1p_1a_2\dots$, we consider what Duplicator would do from the initial configuration of the game $\mathcal{G}^k_{\mathsf{One}}(\mathcal{A},\mathcal{B})$ if Spoiler also moves along r. Suppose Duplicator skips her turn and only moves in some round $n > 0$ by proceeding from the configuration $(p_n, a_1\dots a_n, q_0, \bot, \mathsf{D})$ to

$$(p_n, a_2\dots a_n, q_1, b_1, \mathsf{S}). \tag{3.15}$$

In the game $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A},\mathcal{B})$, for the first $n-1$ rounds, after Spoiler reads a_i, Duplicator moves her pebbles to all of the corresponding a_i-successors, i.e. in every round $i \in \{1,\dots,n-1\}$, she proceeds to the configuration

$$(p_i, S_i, R_i, \mathsf{S}) \tag{3.16}$$

where $S_i = \{q' \mid q \xrightarrow{a_i} q',\, q \in S_{i-1}\}$. By definition of the multi-pebble simulation game, the set R_i contains the positions of pebbles in round i that have not yet seen an accepting state since the last reset. Moreover, by the definition of k', the configuration in (3.16) is valid. For any $i \in \{1, \ldots, n-1\}$, we have $|S_i| \le k'$.

Now in round n, the move of Duplicator is a bit different. She only keeps the pebbles that are generated from the ones that went to q_1 in the first round. Let $S'_1 = \{q_1\}$ and

$$S'_i = \{q' \mid q \xrightarrow{a_i} q',\, q \in S'_{i-1}\}$$

for all $i \in \{2, \ldots, n\}$. In round n, Duplicator proceeds to $(p_n, S'_n, R_n, \mathsf{S})$. The configuration $(p_n, S'_n, R_n, \mathsf{S})$ is also a valid configuration since by definition, $S'_i \subseteq S_i$ for all $i \in \{1, \ldots, n\}$, and hence $|S'_n| \le k'$. Moreover, note that if in (3.15) we have $b_1 = \top$ and in (3.16), for all $i \in \{1, \ldots, n-1\}$, we have $R_i \neq \emptyset$, then the set R_n is empty. This is because all the pebbles in S'_n have gone through q_1 and q_1 is accepting. Hence if in (3.15), $b_1 = \top$, we have $R_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. From the configuration $(p_n, S'_n, R_n, \mathsf{S})$, Duplicator then repeats the same procedure.

Now let $v_0 v'_1 v_1 v'_2 \ldots$ and $u_0 u'_1 u_1 u'_2 \ldots$ be the plays that are obtained in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ and in $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$. Since Duplicator wins the play $v_0 v'_1 v_1 v'_2 \ldots$, either there are infinitely many i such that $v_i = (p_i, \overline{w_i}, q_i, \top, \mathsf{S})$ or only finitely many i such that $v'_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{D})$ with $p_i \in F^{\mathcal{A}}$. The first case implies that we also have infinitely many i such that $u_i = (p_i, S_i, \emptyset, \mathsf{S})$ and the second one implies that there are only finitely many i such that $u'_i = (p_i, a_i, S_i, R_i, \mathsf{D})$ with $p_i \in F^{\mathcal{A}}$. Thus Duplicator also wins the play $u_0 u'_1 u_1 u'_2 \ldots$. She wins $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$. □

This theorem shows that buffered simulation is included in multi-pebble simulation, in the sense that if Duplicator wins the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N} \cup \{\omega\}$ then she also wins the game $\mathcal{G}^{k'}_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ for some k'. One question then is to ask whether the other direction also holds. The answer is negative. Multi-pebble simulation is not included in buffered simulation.

Theorem 3.3.6. *There are two NBA $\mathcal{A}$, $\mathcal{B}$ such that $\mathcal{A} \sqsubseteq^2_{\mathsf{Peb}} \mathcal{B}$, but $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$ for all $k \in \mathbb{N} \cup \{\omega\}$.*

Proof. Consider again the two NBA $\mathcal{A}'$, $\mathcal{B}'$ that are obtained from the automata $\mathcal{A}$, $\mathcal{B}$ from Example 3.1.5 by considering all states to be accepting. Duplicator wins the game $\mathcal{G}^2_{\mathsf{Peb}}(\mathcal{A}', \mathcal{B}')$. The strategy is as follows. Duplicator first moves her two pebbles each to q_b, q_c, and loops there as long as Spoiler reads a. In other words, from the initial configuration $(p_0, \{q_0\}, \emptyset, \mathsf{S})$, after Spoiler proceeds to

$$(p, a, \{q_0\}, \emptyset, \mathsf{D}) \tag{3.17}$$

where $p \in \{p_0, p_1\}$, Duplicator proceeds to

$$(p, \{q_b, q_c\}, \emptyset, \mathsf{S}). \tag{3.18}$$

Now if Spoiler continues by reading a, Duplicator loops in q_b and q_c, and hence proceeds back to (3.18).

If Spoiler eventually reads b or c, Duplicator drops one of her pebbles, duplicates the other one, and moves them to one of the corresponding accepting states. Formally, whenever Spoiler proceeds to $(p_b, b, \{q_b, q_c\}, \emptyset, \mathsf{D})$ or $(p_c, c, \{q_b, q_c\}, \emptyset, \mathsf{D})$ then Duplicator proceeds to

$$(p_b, \{q'_b\}, \emptyset, \mathsf{S}) \text{ or} \tag{3.19}$$
$$(p_c, \{q'_c\}, \emptyset, \mathsf{S}), \tag{3.20}$$

respectively.

There are two possibilities. Either Spoiler eventually reads b or c, or he never does so. In the first case, we eventually reach the configuration (3.19) or (3.20). It is not hard to see that from such configurations, Duplicator can continue accordingly and win the play. In the second case, the play proceeds to the configuration $(p_0, \{q_b, q_c\}, \emptyset, \mathsf{S})$ infinitely often. In this case, Duplicator also wins the play.

Hence Duplicator wins $\mathcal{G}^2_{\mathsf{Peb}}(\mathcal{A}', \mathcal{B}')$. However, we have seen in Corollary 3.1.7 that Duplicator loses the game $\mathcal{G}^{\omega}(\mathcal{A}', \mathcal{B}')$. By Proposition 3.1.6, she also loses the game $\mathcal{G}^{\omega}(\mathcal{A}', \mathcal{B}')$ for all $k \in \mathbb{N}$. Thus we have the desired result. □

3.3.4 Delay Simulation

Regarding delay simulation, there is a strong relation between buffered simulation and delay simulation. Recall that delay simulation is played on an ω-regular language $L \subseteq (\Sigma_1 \times \Sigma_2)^{\omega}$ where both of the players try to construct an infinite word and the play proceeds according to some delay function $f \colon \mathbb{N}^+ \to \mathbb{N}^+$. In round i, Spoiler extends his word with some word over Σ_1 of length $f(i)$ and Duplicator extends her word by reading one letter from Σ_2. The objective of Duplicator is to make the play produce a pair of words that is in L. We have seen in Lemma 2.6.28 that the standard fair simulation game $\mathcal{G}(\mathcal{A}, \mathcal{B})$ is a special case of delay simulation which is played on some ω-regular language $L_{\mathcal{A},\mathcal{B}}$ with a delay function with constant 1. The language $L_{\mathcal{A},\mathcal{B}}$ intuitively expresses the pair of accepting runs of $\mathcal{A}$ and $\mathcal{B}$ that are over the same word.

A buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ where $k \in \mathbb{N}$ can be seen as a special case of delay simulation which is also played on such a language $L_{\mathcal{A},\mathcal{B}}$, but with a delay function with constant $k + 1$. The reduction is basically similar to the one in Lemma 2.6.28.

Theorem 3.3.7. *Given two NBA $\mathcal{A}$, $\mathcal{B}$, there exists an ω-regular language $L_{\mathcal{A},\mathcal{B}}$ such that $\mathcal{A} \sqsubseteq^k \mathcal{B}$ for some $k \in \mathbb{N}$ iff $L_{\mathcal{A},\mathcal{B}}$ is solvable with a delay function f with constant $k + 1$.*

Proof. Given two NBA $\mathcal{A}$, $\mathcal{B}$, consider the language $L_{\mathcal{A},\mathcal{B}}$ as in Lemma 2.6.28. Suppose Duplicator wins the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N}$. By Theorem 3.1.10, she also wins $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$. The winning strategy for Duplicator in the delay simulation game $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$ can be derived from the one in $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$. In the first round of $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$, after Spoiler reads the word $(p_0, a_1, p'_1)(p_1, a_1, p'_2) \ldots (p_k, a_{k+1}, p'_{k+1}) \in E^{\mathcal{A}^*}$, Duplicator does as follows. If there is $i \in \{1, \ldots, k\}$ such that $p'_i \neq p_i$ or p_0 is not an initial state then Duplicator simply chooses an arbitrary transition $(q, a, q') \in E^{\mathcal{B}}$. Otherwise $r = p_0 a_1 p_1 \ldots p'_{k+1}$ is a valid run, and in this case, Duplicator considers what she would do if in the buffered simulation game $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$, Spoiler forms r. Since Duplicator wins $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$, she will move the pebble one step along some transition (q_0, a_1, q_1). Duplicator then reads this transition in the delay simulation game. Duplicator considers a similar procedure for the rest of the play.

Now suppose in $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$, Spoiler forms an infinite word $\rho = (p_0, a_1, p'_1)\,(p_1, a_2, p'_2) \ldots$ and Duplicator forms $\rho' = (q_0, b_1, q'_1)(q_1, b_2, q'_2) \ldots$. Since Duplicator plays according to the winning strategy, whenever $p'_i = p_i$ for all $i \in \mathbb{N}$, and $p_0 a_1 p_1 \ldots$ is an accepting run then $q'_i = q_i$ and $a_i = b_i$ for all $i \in \mathbb{N}$, and $q_0 a_1 q_1 \ldots$ is also an accepting run. By definition, the word $((p_0, a_1, p'_1), (q_0, a_1, q'_1))\ ((p_1, a_2, p'_2), (q_1, a_2, q_2)') \ldots$ is in $L_{\mathcal{A},\mathcal{B}}$. Thus Duplicator wins the play. In the case where $p'_i \neq p_i$ for some $i \in \mathbb{N}$ or $p_0 a_1 p_1 \ldots$ not an accepting run, by the construction of the language $L_{\mathcal{A},\mathcal{B}}$, Duplicator also wins the play. Hence whenever Duplicator wins $\mathcal{G}^k_{\mathsf{Full}}(\mathcal{A}, \mathcal{B})$, she also wins $\mathcal{G}^f_{\mathsf{Del}}(L_{\mathcal{A},\mathcal{B}})$. The other direction can also be shown similarly. □

This theorem shows that we can reduce buffered simulation with one bounded buffer to delay simulation. One may also ask whether the other direction holds, i.e. whether we can reduce delay simulation to buffered simulation with a bounded buffer. The answer is positive. However, we need to consider buffered simulation between two parity automata.

Until now, we have only defined buffered simulation for Büchi automata. Nevertheless, we can also define buffered simulation game for any ω-regular automata similarly. To define such a game formally, we need to modify the third component in the configuration which is used to define the winning condition. In the buffered simulation game for Büchi automata, the third component simply remembers whether Duplicator has seen an accepting state or not. In the buffered simulation game for parity automata, we make the third component remember the highest priority that is seen by Duplicator. The winning condition in buffered simulation over parity automata is essentially the same as the one over Büchi automata. Duplicator wins iff she forms an accepting run or Spoiler does not form an accepting run. We formally define buffered simulation between two parity automata as follows.

Definition 3.3.8. Let $\mathcal{A}$, $\mathcal{B}$ be two parity automata over Σ with priorities $P^{\mathcal{A}}$, $P^{\mathcal{B}} \subseteq \mathbb{N}$ respectively and $k \in \mathbb{N}$. Buffered simulation with one buffer of capacity k between $\mathcal{A}$, $\mathcal{B}$ is $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are

$$\begin{aligned} V_1 &= Q^{\mathcal{A}} \times \Sigma^{\leq k} \times Q^{\mathcal{B}} \times P^{\mathcal{B}} \cup \{0\} \times \{\mathsf{S}\}, \\ V_0 &= Q^{\mathcal{A}} \times \Sigma^{\leq k} \times Q^{\mathcal{B}} \times P^{\mathcal{A}} \times \{\mathsf{D}\}, \end{aligned}$$

$V = V_0 \cup V_1$, the edge relation E is defined as

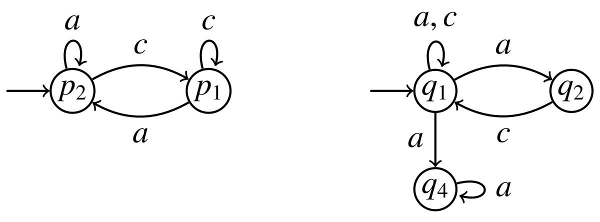
$$\begin{aligned} (p, w, q, b, \mathsf{S}) \to (p', wa, q, b, \mathsf{D}) \text{ in } E \quad &\text{iff} \quad p \xrightarrow{a} p', \\ (p, w, q, b, \mathsf{D}) \to (p, w', q_n, b', \mathsf{S}) \text{ in } E \quad &\text{iff} \quad \exists u = a_1 \ldots a_n \in \Sigma^* \text{ such that} \\ & \qquad q \xrightarrow{a_1} q_1 \ldots \xrightarrow{a_n} q_n,\ w = uw', \text{ and} \\ & \qquad b' = \begin{cases} \max\{\Omega^{\mathcal{B}}(q_1), \ldots, \Omega^{\mathcal{B}}(q_n)\} & \text{if } u \neq \epsilon \\ 0 & \text{else,} \end{cases} \end{aligned}$$

the initial configuration is $v_0 = (p_0, \epsilon, q_0, 0, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}, \mathcal{B}$, and the play

$$(p_0, w_0, q_0, b_0, \mathsf{S})(p_1, w'_0, q_0, b_0, \mathsf{D})(p_1, w_1, q_1, b_1, \mathsf{S})(p_2, w'_1, q_1, b_1, \mathsf{D}) \ldots$$

is in Win iff $\mathsf{max}(\mathsf{inf}(b_0 b_1 \ldots))$ is even or $\mathsf{max}(\mathsf{inf}(\Omega^{\mathcal{A}}(p_0)\Omega^{\mathcal{A}}(p_1) \ldots))$ is not even. We write $\mathcal{A} \hat{\sqsubseteq}^k \mathcal{B}$ if Duplicator wins the game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$.

Example 3.3.9. Consider the following two parity automata $\mathcal{A}$, $\mathcal{B}$ where $\Omega^{\mathcal{A}}(p_i) = \Omega^{\mathcal{B}}(q_i) = i$.



We have $\mathcal{A} \not\hat{\sqsubseteq}^k \mathcal{B}$ for any $k \in \mathbb{N}$ since in the game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$, Spoiler has the following winning strategy. He loops in the initial state p_2 by reading a until Duplicator goes to q_4, i.e. from any configuration

$$(p_2, w, q, b, \mathsf{S}), \tag{3.21}$$

if $q \neq q_4$, Spoiler proceeds to $(p_2, wa, q, b, \mathsf{D})$.

If Duplicator eventually goes to q_4 then Spoiler goes to p_1 and loops there for the rest of the play by reading c. Formally, if we reach a configuration as in (3.21) with $q = q_4$, Spoiler proceeds to $(p_1, wc, q_4, b, \mathsf{S})$ and from any configuration $(p_1, wc, q_4, b, \mathsf{S})$, Spoiler proceeds to $(p_1, wcc, q_4, b, \mathsf{S})$. Duplicator eventually gets stuck from such a configuration since there is no c-transition from q_4.

However, if Duplicator never goes to q_4, there are two cases: either she eventually goes to q_2 or she never leaves the initial state. In the second case, Spoiler also wins since he forms an accepting run while Duplicator does not, i.e. we obtain a play $(p_0, w_0, q_0, b_0, \mathsf{S})$ $(p_1, w'_0, q_0, b_0, \mathsf{D})$ $(p_1, w_1, q_1, b_1, \mathsf{S}) \dots$ where $\mathsf{max}(\mathsf{inf}(b_0 b_1 \dots))$ is 1, but $\mathsf{max}(\mathsf{inf}(\Omega^{\mathcal{A}}(p_0)\Omega^{\mathcal{B}}(p_1)\dots))$ is 2. In the first case, we reach some configuration $(p_2, w, q_2, b, \mathsf{D})$ where $w \in a^+$. Since from q_2 there is no a-transition, from such a configuration, Duplicator eventually gets stuck. Hence Spoiler wins the play. He wins $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$ for any $k \in \mathbb{N}$.

We can reduce delay simulation to buffered simulation between two parity automata in which one of them is deterministic. Recall that deterministic parity automata (DPA) are as expressive as NBA. They both recognise exactly the class of ω-regular languages [Saf88, Tho90]. Given an ω-regular language $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, let C be a deterministic parity automaton that recognises L. We will consider buffered simulation between two parity automata $\mathcal{A}$, $\mathcal{B}$, in which $\mathcal{A}$ is deterministic and recognises $L_1 \subseteq \Sigma_1^\omega$, the projection of L to Σ_1, and $\mathcal{B}$ is non-deterministic and obtained from C by remembering the second component of the letters in its state space. We formally show the reduction as follows.

Theorem 3.3.10. *Given an ω-regular language $L \subseteq \Sigma_1 \times \Sigma_2$, there are two parity automata $\mathcal{A}$, $\mathcal{B}$ such that $\mathcal{A} \sqsubseteq^k \mathcal{B}$ for some $k \in \mathbb{N}$ iff L is solvable with some delay function f with constant $k+1$.*

Proof. Let $C = (Q, \Sigma_1 \times \Sigma_2, E, q_0, \Omega)$ be a deterministic parity automaton such that $L(C) = L$. We construct two parity automata $\mathcal{A}$, $\mathcal{B}$ over Σ_1 as follows. The automaton $\mathcal{A}$ is a deterministic parity automaton that recognises the language

$$L_1 = \{a_1 a_2 \dots \in {\Sigma_1}^\omega \mid \exists b_1 b_2 \dots \in {\Sigma_2}^\omega : (a_1, b_1)(a_1, b_2) \dots \in L\}. \tag{3.22}$$

There is such a deterministic parity automaton $\mathcal{A}$ since the language L is regular, and hence L_1 is also regular. The automaton $\mathcal{B}$ is the tuple $(Q \times (\Sigma_2 \cup \{\epsilon\}), \Sigma_1, E', (q_0, \epsilon), \Omega')$ where E' is defined as

$$((p, b'), a, (p', b)) \in E' \text{ iff } (p, (a, b), p') \in E$$

and the priority function Ω' is defined as $\Omega'(p, b) = \Omega(p)$ for all $(p, b) \in Q \times (\Sigma_2 \cup \{\epsilon\})$.

Now suppose Duplicator wins the game $\mathcal{G}^f_{\mathsf{Del}}(L)$ with some delay function f with constant $k+1$. In the game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$, Duplicator wins with the following winning strategy: in the first k rounds, she skips her turn. Hence in round $k+1$, the play proceeds to some configuration $(p_{k+1}, a_1 \dots a_{k+1}, (q_0, \epsilon), 0, \mathsf{D})$. In this case, Duplicator then considers what she would do in the first round of $\mathcal{G}^f_{\mathsf{Del}}(L)$ assuming that Spoiler reads $a_1 \dots a_{k+1}$. If Duplicator reads b_1 then she considers the state q_1 in C that can be reached by reading (a_1, b_1) from q_0. There is such a unique q_1 since C is deterministic. In the buffered simulation game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$, Duplicator proceeds to $(p_{k+1}, a_2 \dots a_{k+1}, (q_1, b_1), \Omega(q_1), \mathsf{S})$, i.e. she moves the pebble from the initial state (q_0, ϵ) to the state (q_1, b_1) by reading a_1. For the rest of the play, Duplicator considers the same procedure: if in round i of the delay simulation game $\mathcal{G}^f_{\mathsf{Del}}(L)$, Duplicator extends her word with b_i then in round $i+k$ of the game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$, from the state (q_{i-1}, b_{i-1}), Duplicator extends her run by reading a_i and going to the state (q_i, b_i) where q_i is the state that can be reached in C by reading (a_i, b_i) from q_{i-1}.

Suppose in the game $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$, Spoiler forms an accepting run over some word $a_1 a_2 \dots \in L_1$. By the construction of the winning strategy, Duplicator forms a run

$$\rho' = (q_0, \epsilon) \xrightarrow{a_1} (q_1, b_1) \xrightarrow{a_2} \dots$$

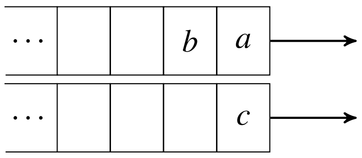
over $a_1 a_2 \dots$. By the definition of $\mathcal{B}$, there is a corresponding run $\rho = q_0(a_1, b_1)\, q_1(a_2, b_2) \dots$ in C. Since C is deterministic, such a run ρ over $w = (a_1, b_1)(a_2, b_2) \dots$ is unique. Moreover $w \in L$ since Duplicator wins $\mathcal{G}^f_{\mathsf{Del}}(L)$. Since w is accepted by C, the run ρ is accepting. This implies that ρ' is also accepting. Hence Duplicator wins the play. Duplicator wins $\hat{\mathcal{G}}^k(\mathcal{A}, \mathcal{B})$ if she wins $\mathcal{G}^f_{\mathsf{Del}}(L)$. The other direction can also be shown similarly. □

Together with Theorem 3.3.7, this theorem shows that to some extent, delay simulation is equivalent to buffered simulation with one bounded buffer. The capacity of the buffer corresponds to the delay constant. However, note that in the case of unbounded buffer, the buffered simulation game cannot be reduced to any delay simulation game. The reason is simply because there is no corresponding constant for the unbounded capacity.

3.4 Simulation with $n \geq 1$ Buffers

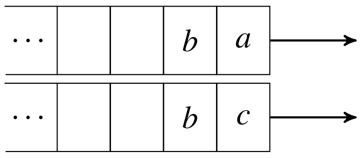
We can naturally extend the definition of buffered simulation in Definition 3.1.1 to a more general case, where instead of only one buffer, we have several buffers. Duplicator can use multiple buffers to store Spoiler's letters. However, unlike in the case where we only have one buffer, in this case, every time Spoiler reads a letter, we also have to determine to which buffer the letter should be put. There are several possibilities to define this. For example, we can consider a game where one of the players decides this or we can also consider a game where there is a fixed rule that tells to which buffer the letter should be put. In this work, we will consider a game where each letter determines the buffers where it should be put.

The motivation behind this choice comes from the area of formal languages. First note that the possibility to put letters to different buffers is strongly related to the correspondence between the words that are produced by Spoiler and Duplicator. If there is only one buffer then Duplicator has no choice except to produce exactly the same word that is produced by Spoiler. However if there is more than one buffer then Duplicator may produce a different word than Spoiler. For instance, suppose we have two buffers in which the letters a and b should be put into the first buffer and the letter c into the second one. If now Spoiler reads the word abc and Duplicator skips her turns then the condition of the buffers is as follows.



The first buffer is filled with a, b and the second one with c. Duplicator can respond to this by first popping the first buffer and then the second one. In such a case, she reads abc. However, she can also pop the buffers with a different order: she first pops one letter from the first buffer, then the second one, and then the first one again. In such a case, she reads acb. The fact that a and b get stored in the same buffer introduces a dependency between the letters a and b. Duplicator can only read a word in which a and b occur in the same order as in the Spoiler's word. On the other hand, the fact that a and b get stored in a different buffer than c introduces an independency between the letters a, b and the letter c with respect to the order in which they occur in Spoiler's and Duplicator's words. This is intuitively the reason why after Spoiler reads abc, Duplicator may respond with abc, acb, or even cab, i.e. by popping the two letters from the second buffer and then the first buffer. The words abc, acb, and cab are considered to be equivalent to abc modulo to such an independency. This equivalence is already known in the area of formal language by the name of *(Mazurkiewicz) traces* [Maz89]. We will discuss more about Mazurkiewicz traces together with the application of buffered simulation in such area later in Chapter 5,

In general, the dependency between the letters is not necessarily transitive. For instance, we might have a case where the letters a, b are dependent on each other, b, c are also dependent on each other, but a, c are independent of each other. If we want to model such a dependency then intuitively the letters a and c have to be stored in two different buffers, but b in the same buffer as a, as well as, c. Thus in buffered simulation, we will allow a letter to be put into several buffers. For example, to model such a case, we consider a game with two buffers where the letter a is stored in the first buffer, c in the second one, and b in both the first and the second buffers. Whenever Spoiler reads b then two copies of b are pushed each to the first and second buffers, and if Duplicator wants to read b then b has to be popped from the first and second buffers. Duplicator cannot read b if b is not in the top of the first and second buffers. To illustrate this, suppose Spoiler reads acb and Duplicator skips her turns. The condition of the buffers then is as follows.



The first buffer is filled with a, b and the second one with c, b. Duplicator then can respond to this by first popping a from the first buffer, c from the second one, and then b from both of the first and the second buffers. In such a case, Duplicator will read acb. However, she can also pop c and a in the reverse direction. She first pops c from the first buffer, a from the second one, and then followed by popping b from the first and the second buffers. In such a case, Duplicator reads *cab*. These, however, are the only possibilities for Duplicator. Duplicator, for instance, cannot pop a and then b since once she pops b, she has to pop it from both the first and second buffers.

We will formally define buffered simulation with multiple buffers, by using the notion of *distributed alphabet* [Zie87]. A distributed alphabet basically is a tuple of (not necessarily disjoint) alphabets $(\Sigma_1, \ldots, \Sigma_n)$. We will simply consider two NBA $\mathcal{A}$, $\mathcal{B}$ over a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ instead of an ordinary alphabet Σ. Intuitively the NBA $\mathcal{A}$, $\mathcal{B}$ are defined over the alphabet $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ and for all $i \in \{1, \ldots, n\}$, the alphabet Σ_i tells us the set of letters that are stored to the i-th buffer. We will assume that the distributed alphabet is part of the input.

In the buffered simulation game with multiple buffers, each of the buffers might have a different capacity. One buffer might be bounded and the other one might be unbounded. Hence instead of considering a single capacity, we will use a *capacity vector* to denote the capacity of the buffers. More precisely, we will define the game on two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ with respect to some capacity vector $\kappa = (k_1, \ldots, k_n) \in (\mathbb{N} \cup \{\omega\})^n$. For all $i \in \{1, \ldots, n\}$, the capacity k_i is the capacity of the i-th buffer.

A buffered simulation game played on two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ with capacity $\kappa = (k_1, \ldots, k_n)$ proceeds as follows. In each round, Spoiler moves the pebble in $\mathcal{A}$ one step by reading a letter a and pushes a copy of a to all buffers $i \in \{1, \ldots, n\}$ where $a \in \Sigma_i$, i.e. to all buffers that are associated with a. Duplicator can either skip her turn or move the pebble in $\mathcal{B}$ by reading $b_1 \ldots b_m$. If she reads $b_1 \ldots b_m$ then for each b_j, started from $j = 1$, she pops b_j from all buffer $i \in \{1, \ldots, n\}$ where $b_j \in \Sigma_i$, i.e. from all buffers that are associated with b_j. The winning condition is the same as the game with one buffer. Duplicator wins if either Spoiler gets stuck or whenever Spoiler forms an accepting run, Duplicator also forms an accepting run. Duplicator also has to obey the capacity restriction. She loses immediately if she violates the capacity restriction of one of the buffers. Furthermore, we additionally require that every letter that is pushed to the buffers has to be eventually popped by Duplicator. This is to make sure that Duplicator plays fairly. She should not win the game if she only pops letters from some certain buffers and ignores other buffers.

Before we give the formal definition of buffered simulation with multiple buffers, let us consider the notion of *projection*. The projection of a word w over Σ to $\Sigma_i \subseteq \Sigma$ is basically a word that is obtained from w by deleting all letters that do not belong to Σ_i. We will use the notion of projection to denote the content of some buffer.

Definition 3.4.1. Given a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. For any $i \in \{1, \ldots, n\}$, a projection π_i is a function $\pi_i : \Sigma^\infty \to \Sigma_i^\infty$ such that $\pi_i(a_1 a_2 \ldots) = \pi_i(a_1)\pi_i(a_2)\ldots$ where $\pi_i(a) = a$ if $a \in \Sigma_i$ and $\pi_i(a) = \epsilon$ if $a \notin \Sigma_i$.

For example, suppose we have a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ in which $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a, c\}$. The projections of $w = ababc$ to Σ_1 and Σ_2 are respectively $\pi_1(w) = abab$ and $\pi_2(w) = aac$.

We formally define the buffered simulation game with multiple buffers as follows.

Definition 3.4.2. Given two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a capacity vector $\kappa = (k_1, \ldots, k_n) \in (\mathbb{N} \cup \{\omega\})^n$, *buffered simulation game* denoted with $\mathcal{G}^\kappa_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, or simply $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$, is a tuple $((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times (\Sigma_1^{\le k_1} \times \ldots \times \Sigma_n^{\le k_n}) \times Q^{\mathcal{B}} \times \{1, \ldots, n, \top, \bot\} \times \{\mathsf{S}\},$$

$$V_0 = Q^{\mathcal{A}} \times (\Sigma_1^{\leq k_1+1} \times \ldots \times \Sigma_n^{\leq k_n+1}) \times Q^{\mathcal{B}} \times \{1, \ldots, n, \top, \bot\} \times \{\mathsf{D}\},$$

where $\Sigma_i^{\leq k_i} = \Sigma_i^{\leq k_i+1} = \Sigma^*$ if $k_i = \omega$. Let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$, the edge relation E is defined as

$$(p, (w_1, \ldots, w_n), q, c, \mathsf{S}) \rightarrow (p', (w_1\pi_1(a), \ldots, w_n\pi_n(a)), q, c, \mathsf{D}) \text{ in } E$$
$$\text{iff} \quad p \xrightarrow{a} p'$$

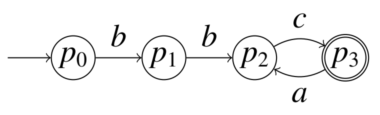
$$(p, (w_1, \ldots, w_n), q, c, \mathsf{D}) \rightarrow (p, (w_1', \ldots, w_n'), q', c', \mathsf{S}) \text{ in } E$$
$$\exists u = a_1 \ldots a_m \in \Sigma^* \text{ such that } q \xrightarrow{u} q'$$
$$w_i = \pi_i(u)w_i' \text{ for all } i \in \{1, \ldots, n\}, \text{ and}$$
$$\text{iff} \quad c' = \begin{cases} 1 & \text{if } c = \top, \\ c+1 & \text{if } c \in \{1, \ldots, n-1\} \text{ and either } w_c = \epsilon \text{ or },\\ & \exists j \in \{1, \ldots, m\} : a_j \in \Sigma_n, \\ \bot & \text{if } c = n \text{ and either } w_n = \epsilon \text{ or } \exists j \in \{1, \ldots, m\} : a_j \in \Sigma_c, \\ \top & \text{if } c = \bot, u \neq \epsilon, \text{ and } q \xrightarrow[F]{u} q', \\ c & \text{otherwise,} \end{cases}$$

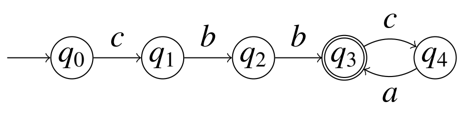
the initial configuration is $v_0 = (p_0, \epsilon, q_0, \bot, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and we have $v_0v_1 \ldots \in \mathsf{Win}$ iff there exist infinitely many i such that $v_i = (p, w, q, \top, \mathsf{S})$ or there are only finitely many i such that $v_i = (p, w, q, b, \mathsf{D})$ with $p \in F^{\mathcal{A}}$. We write $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ if Duplicator wins $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$.

The definition of $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ is similar to the one in Definition 3.1.1. The first and the third components of the configuration represent the positions of Spoiler's and Duplicator's pebbles and the second one represents the content of the buffer. In this case, instead of a single word, the second component is an n-tuple of words $(w_1, \ldots, w_n)$. For all $i \in \{1, \ldots, n\}$, the word w_i represents the content of the i-th buffer. The last component of the configuration tells us the player that has the next turn and the second last one is used to determine the winning condition. In this case, instead of a bit, it is a counter. The counter starts from $\bot$. It changes to $\top$ if Duplicator moves her pebble through an accepting state, and from $\top$, it changes to 1. From 1, the counter increases gradually to n. Intuitively, the counter increases from c to $c + 1$ if Duplicator pops at least one letter from buffer c or buffer c is empty. If the counter is n and Duplicator pops a letter from buffer n or buffer n is empty then the counter resets to $\bot$. Thus if the counter reaches $\top$ infinitely often, this means that Duplicator never ignores a buffer and sees an accepting state infinitely often.

As in the case of buffered simulation with one buffer, the requirement where Duplicator should obey the capacity restriction is included in the definition of valid configurations. Spoiler's configuration $(p, (w_1, \ldots, w_n), q, c, \mathsf{S})$ is only valid if $w_i \leq k_i$ for all $i \in \{1, \ldots, n\}$. Hence Duplicator can only proceed to a configuration where each buffer i contains at most k_i many letters. If such a move is not possible then Duplicator gets stuck and loses the play immediately. Since we only check the capacity restriction after Duplicator's move, the configuration $(p, (w_1, \ldots, w_n), q, c, \mathsf{D})$ is valid if $w_i \leq k_i + 1$ for all $i \in \{1, \ldots, n\}$. Spoiler can push one more letter into "full" buffers and it is Duplicator's responsibility to shorten the buffers again. Hence we can also consider buffered simulation in which some buffers have capacity 0. If buffer $i \in \{1, \ldots, n\}$ has a capacity 0 then this just means that Duplicator has to pop the letter that is pushed into buffer i immediately.

Example 3.4.3. Consider a buffered simulation game that is played in the following two NBA $\mathcal{A}$, $\mathcal{B}$ over the distributed alphabet $\hat{\Sigma} = (\{a, b\}, \{b\}, \{c\})$ and a capacity $\kappa = (2, 2, 0)$.

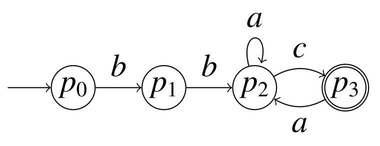


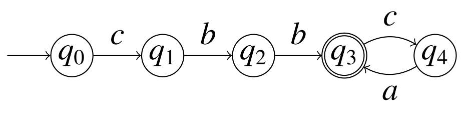


In the game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$, there are three buffers where the letter a will be stored in the first one, b in the first and the second ones, and c in the third one. Duplicator wins the game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ with the following winning strategy. Initially, she skips her turn in the first two rounds. Hence at the end of the first round, after Spoiler reads b and Duplicator skips her turn, we reach the configuration $(p_1, (b, b, \epsilon), q_0, \bot, \mathsf{S})$. In the second round, after Spoiler reads another b and Duplicator skips her turn, we reach the configuration $(p_1, (bb, bb, \epsilon), q_0, \bot, \mathsf{S})$. In the third round, Spoiler will read c and proceed to the configuration $(p_3, (bb, bb, c), q_0, \bot, \mathsf{D})$. In this case, Duplicator empties the buffers by reading cbb and going to the accepting state q_3, i.e. she proceeds to the configuration $(p_3, (\epsilon, \epsilon, \epsilon), q_3, \top, \mathsf{S})$. In the next round, Spoiler will read a and Duplicator cannot do anything except to skip her turn, i.e. she proceeds to the configuration $(p_2, (a, \epsilon, \epsilon), q_3, 1, \mathsf{S})$. Spoiler then will read c and in this case, Duplicator pops both c and a from the buffers and goes back to q_3 by reading ca. The play then proceeds to the configuration $(p_3, (\epsilon, \epsilon, \epsilon), q_3, 2, \mathsf{S})$. Duplicator repeats this procedure for the rest of the play: she waits in q_3 until Spoiler pushes a and c to the buffers, and then goes back to q_3 by emptying the buffers. The counter resets infinitely often and we will reach the configuration $(p_3, (\epsilon, \epsilon, \epsilon), q_3, \top, \mathsf{S})$ infinitely often. By definition, Duplicator wins the play.

As in the case of buffered simulation with one buffer, we can also consider a game where some buffers have an unbounded capacity.

Example 3.4.4. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\{a, b\}, \{b\}, \{c\})$ that are obtained from Example 3.4.3 by adding an a-loop from p_2.





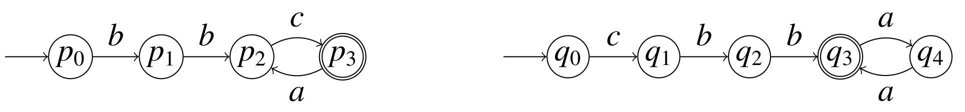
First note that to win a buffered simulation game that is played on such NBA, Duplicator needs an unbounded buffer to store unboundedly many a. Duplicator loses any game $\mathcal{G}^{k_1,k_2,k_3}(\mathcal{A}, \mathcal{B})$ if $k_1 \in \mathbb{N}$ since Spoiler has the following winning strategy. He first reads bb and then loops in p_2 by reading a for the rest of the play. The play eventually reaches the configuration $(p_2, (bba^{k_1-1}, bb, \epsilon), q_0, \bot, \mathsf{D})$ which does not have a valid successor. Hence Duplicator loses the play.

Duplicator, however, wins the game $\mathcal{G}^{\omega,2,0}(\mathcal{A}, \mathcal{B})$. Intuitively, she skips her turn and moves her pebble only when Spoiler reads c. Note that Spoiler will read c by going to p_3, infinitely often, since otherwise he will lose for not forming an accepting run, i.e. we will obtain a play $v_0 v_1 \ldots$ where there are only finitely many i such that $v_i = (p, w, q, b, \mathsf{D})$ with $p \in F^{\mathcal{A}}$.

Now suppose Spoiler reads his first c after reading a, $m_1 \geq 0$ many times. We reach the configuration $(p_3, (bba^{m_1}, bb, c), q_0, \bot, \mathsf{D})$. In this case, Duplicator empties the second and the third buffers by going to the accepting state q_3 i.e. she proceeds to the configuration $(p_3, (a^{m_1}, \epsilon, \epsilon), q_3, \top, \mathsf{S})$. Duplicator then waits again until Spoiler reads c, i.e. until the play proceeds to some configuration $(p_3, (a^{m_1+m_2+1}, \epsilon, c), q_3, 1, \mathsf{D})$ where $m_2 \geq 0$. In such a case, Duplicator empties the last two buffers again. She proceeds to $(p_3, (a^{m_1+m_2}, \epsilon, \epsilon), q_3, 2, \mathsf{S})$. We can repeat this principle for the rest of the play. The counter resets infinitely often and we will reach some configuration $(p_3, (w, \epsilon, \epsilon), q_3, \top, \mathsf{S})$ where $w \in a^*$, infinitely often. Duplicator wins such a play. She wins $\mathcal{G}^{\omega,2,0}(\mathcal{A}, \mathcal{B})$.

In this example, the content of the first buffer in the game $\mathcal{G}^{\omega,2,0}(\mathcal{A}, \mathcal{B})$ might keep growing since Duplicator can pop at most one a each time she moves, but Spoiler can push several as before Duplicator moves. However, Duplicator still plays fairly since every a that is pushed to the buffer is eventually popped by Duplicator. In the following, we show an example where Duplicator can only form an accepting run by playing unfairly.

Example 3.4.5. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\{a, b\}, \{b\}, \{c\})$ that are obtained by slightly modifying Duplicator's automaton in Example 3.4.3. We change the label of the outgoing edge from q_3 from c to a.



$$\begin{tikzcd} \arrow[r] & p_0 \arrow[r, "b"] & p_1 \arrow[r, "b"] & p_2 \arrow[r, bend left, "c"] & p_3 \arrow[l, bend left, "a"] \end{tikzcd} \qquad \begin{tikzcd} \arrow[r] & q_0 \arrow[r, "c"] & q_1 \arrow[r, "b"] & q_2 \arrow[r, "b"] & q_3 \arrow[r, bend left, "a"] & q_4 \arrow[l, bend left, "a"] \end{tikzcd}$$

In this case, Spoiler wins the game $\mathcal{G}^{\omega,\omega,\omega}(\mathcal{A}, \mathcal{B})$ by reading the word $bb(ca)^\omega$. First note that if Spoiler reads such a word, Duplicator loses if she does not move infinitely often since Spoiler forms an accepting run. Hence let us assume that Duplicator moves infinitely often. In the first two rounds, Duplicator cannot move her pebble because there is only c-transition from the initial state. However, in the third round, after Spoiler reads c, we reach the configuration $(p_3, (bb, bb, c), q_0, \bot, \mathsf{D})$. Duplicator might pop everything from the buffer and go to the accepting state, i.e. she proceeds to the configuration $(p_3, (\epsilon, \epsilon, \epsilon), q_3, \top, \mathsf{S})$. From this configuration, after Spoiler reads $acac$ and proceeds to $(p_3, (aa, \epsilon, cc), q_3, 1, \mathsf{D})$, Duplicator might continue by reading aa and going back to the accepting state. Hence we reach the configuration $(p_3, (\epsilon, \epsilon, cc), q_3, 2, \mathsf{S})$. Duplicator then can repeat this procedure for the rest of the play and successively proceed to the configurations $(p_3, (\epsilon, \epsilon, cccc), q_3, 3, \mathsf{S})$, $(p_3, (\epsilon, \epsilon, cccccc), q_3, 3, \mathsf{S})$ etc., each time after Spoiler reads $acac$. Duplicator's pebble visits an accepting state infinitely often. However the play never reaches a configuration in which the counter is $\top$ because after the third round, Duplicator never pops c from the third buffer. Duplicator loses the play by considering such a strategy. In fact, there is no way for Duplicator to pop the second c that is pushed to the buffer. She loses the game $\mathcal{G}^{\omega,\omega,\omega}(\mathcal{A}, \mathcal{B})$.

In the following, we give another example in which Duplicator loses the buffered simulation game. We consider the same NBA $\mathcal{A}$, $\mathcal{B}$ as in Example 3.4.4, but with a slightly different distributed alphabet.

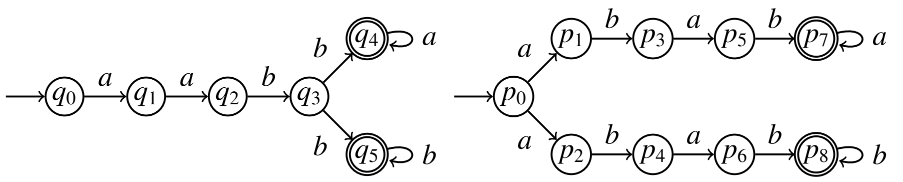
Example 3.4.6. Consider again the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 3.4.4, but now over a distributed alphabet $\hat{\Sigma} = (\{a, b\}, \{b, c\})$. Hence we will consider a game with two buffers in which the letter a is pushed to the first one, b to the first and the second ones, and c to the second one. In this case, we have $\mathcal{A} \not\sqsubseteq^{\omega,\omega} \mathcal{B}$. Spoiler wins the game $\mathcal{G}^{\omega,\omega}(\mathcal{A}, \mathcal{B})$ by reading $bb(ca)^\omega$. In the first three rounds, Duplicator cannot do anything except to skip her turn and hence in the third round, we reach the configuration

$$(p_3, (bb, bbc), q_0, \bot, \mathsf{D}). \tag{3.23}$$

However from this configuration, Duplicator also cannot pop the buffer since q_0 only has a c-transition and c is not at the top of the second buffer. From configuration (3.23), Duplicator can only skip her turn for the rest of the play. Hence we obtain a play $v_0 v_1 \ldots$ in which there is no i such that $v_i = (p, \overline{w}, q, \top, \mathsf{S})$, but there are infinitely many i such that $v_i = (p, \overline{w}, q, b, \mathsf{D})$ where $p \in F^{\mathcal{A}}$. Thus Spoiler wins.

We can also consider the flushing variant of buffered simulation with multiple buffers in the same way as in the case of one buffer. Recall that the flushing variant is a more restricted case of buffered simulation in which Duplicator has to pop the entire content of the buffers once she decided to move her pebble. The formal definition of the flushing variant of buffered simulation with multiple buffers only differs from Definition 3.4.2 in the length of the word u that is chosen by Duplicator. We have $|u| = 0$ or $|u| = |w|$ where w is the word that satisfies $\pi_i(w) = w_i$ for all $i \in \{1, \ldots, n\}$. Hence by reading w, Duplicator empties all the buffers. For any two NBA $\mathcal{A}$, $\mathcal{B}$ over a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a capacity vector $\kappa = (k_1, \ldots, k_n)$, we denote the flushing variant between two NBA $\mathcal{A}$, $\mathcal{B}$ by $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$. We write $\mathcal{A} \sqsubseteq^{\kappa}_{\mathsf{Flush}} \mathcal{B}$ if Duplicator wins the flushing variant $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$.

Example 3.4.7. Consider the following two NBA $\mathcal{A}$ and $\mathcal{B}$ over $\hat{\Sigma} = (\{a\}, \{b\})$.



Duplicator wins the game $\mathcal{G}^{2,1}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ with the following winning strategy. Initially, she skips her turn in the first three rounds. Hence at the end of the third round, we reach the configuration $(q_3, (aa, b), p_0, \perp, \mathsf{S})$. In the next round, after Spoiler proceeds to $(q_4, (aa, bb), p_0, \perp, \mathsf{D})$ or $(q_5, (aa, bb), p_0, \perp, \mathsf{D})$ by reading b, Duplicator responds to this by reading $abab$ and empties the buffer. She proceeds to $(q_4, (\epsilon, \epsilon), p_7, \top, \mathsf{S})$ or $(q_5, (\epsilon, \epsilon), p_8, \top, \mathsf{S})$ respectively. From such configurations, it is not hard to see that Duplicator can continue accordingly and win the play.

In a buffered simulation game with one buffer, restricting Duplicator to move only at most one step in each round does not make any difference to the general case where Duplicator is allowed to move several steps at each round. This, however, is not the case if we consider a game with multiple buffers. There are some cases in which Duplicator can only win if she is allowed to move more than one step in each round. To show this formally, let $\mathcal{G}^{\kappa}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ be a buffered simulation game between two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and capacity $\kappa = (k_1, \ldots, k_n)$ where Duplicator is restricted to pop at most one letter in each round. Hence the formal definition of $\mathcal{G}^{\kappa}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ is the same as in Definition 3.4.2, but with $|u| \leq 1$. In the following, we show that there are two NBA $\mathcal{A}$, $\mathcal{B}$ in which Duplicator wins $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$, but loses $\mathcal{G}^{\kappa}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$.

Example 3.4.8. Consider the two NBA $\mathcal{A}$ and $\mathcal{B}$ over $\hat{\Sigma} = (\{a\}, \{b\})$ from Example 3.4.7. We have seen that Duplicator wins the game $\mathcal{G}^{2,1}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, and hence also wins the game $\mathcal{G}^{2,1}(\mathcal{A}, \mathcal{B})$ with the same winning strategy. She, however, loses the game $\mathcal{G}^{2,1}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$. Spoiler has a winning strategy by first reading aab.

If Duplicator skips her turn in the first three rounds then the play proceeds to the configuration $(q_3, (aa, b), p_0, \perp, \mathsf{S})$. In the next round, Spoiler proceeds to the configuration

$(q_4, (aa, bb), p_0, 1, \mathsf{D})$. Such a configuration has no valid successor in $\mathcal{G}^{2,1}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$ since Duplicator can only pop one a from the buffer, and this would leave the second buffer with two letters. Hence from such a configuration, Duplicator gets stuck and loses the play.

Now let us assume that Duplicator moves during the first three rounds. At the end of the third round, the play proceeds to some configuration $(q_3, (w_1, w_2), p_i, \bot, \mathsf{S})$ where $i \in \{1, \ldots, 6\}$. From such a configuration, Spoiler then proceeds to the configuration $(q_4, (w_1, w_2 b), p_i, \bot, \mathsf{D})$ if i is even and to $(q_5, (w_1, w_2 b), p_i, \bot, \mathsf{D})$ if i is odd. In the first case, the play eventually proceeds to some configuration $(q_4, (w, \epsilon), p_8, \top, \mathsf{D})$ where $w \in a^+$, and in the second one, to $(q_5, (\epsilon, w), p_7, \top, \mathsf{D})$ where $w \in b^+$. Since from p_7, there is no b-transition and from p_8, there is no a-transition, such configurations do not have a valid successor. Hence Duplicator loses the game $\mathcal{G}^{2,1}_{\mathsf{One}}(\mathcal{A}, \mathcal{B})$.

In the following section, we will characterise the expressive power of buffered simulation. We will see how expressive buffered simulation is with respect to the distributed alphabets and capacity vectors. We will also see some possibilities of reduction between buffered simulation itself.

3.5 Expressive Power

Before we show the expressive power of buffered simulation with multiple buffers, first note that there is no particular order on the tuples in the distributed alphabet. The distributed alphabet simply tells us how the letters are distributed among the buffers. For instance, the distributed alphabet $\hat{\Sigma} = (\{a, b\}, \{b, c\})$ and $\hat{\Sigma}' = (\{b, c\}, \{a, b\})$ intuitively tells us the same thing: there are two buffers in which one of them is used to store a, b, and the other one for b, c.

However, if we consider a pair of distributed alphabet and a capacity vector, there is a natural order in which they are written as a pair of n-tuples. For example, consider again the previous distributed alphabets $\hat{\Sigma}$, $\hat{\Sigma}'$, and two distinct numbers k_1, k_2. The games $\mathcal{G}^{k_1,k_2}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{k_1,k_2}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ are clearly different. In the first one, we can store at most k_1 many as and in the second one k_2 many as. The games $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ are equivalent if the pair of the distributed alphabet and capacity vectors $\hat{\Sigma}', \kappa'$ is a permutation of $\hat{\Sigma}, \kappa$. In such a case, if Duplicator wins $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, she also wins the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. She simply moves her pebble in the same way as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$.

One important observation is that if during a play in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator resets the counter once, it is not necessary that she also resets the counter once in the corresponding game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. The order of the buffers is different. For instance, in some round of $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator might pop a letter from the first buffer and increase the counter from 1 to 2, but in the corresponding game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, the letter might be popped from the last buffer and we do not increase the counter if it is 1. However, since in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator eventually will pop the letter from the j-th buffer, where $\sigma(j) = 1$, in the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, the counter eventually will also increase from 1 to 2. Duplicator resets the counter infinitely often in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ if she also does so in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. This is not an overwhelming finding in the theory of buffered simulation but stating this observation allows us to formulate some results more easily.

Theorem 3.5.1. *Let* $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ *be a distributed alphabet,* $\kappa = (k_1, \ldots, k_n)$ *a capacity vector, and* $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ *a permutation. Let* $\hat{\Sigma}' = (\Sigma_{\sigma(1)}, \ldots, \Sigma_{\sigma(n)})$ *and*

$\kappa' = (k_{\sigma(1)}, \ldots, k_{\sigma(n)})$. We have

$$\sqsubseteq^{\kappa}_{\hat{\Sigma}} = \sqsubseteq^{\kappa'}_{\hat{\Sigma}'} .$$

Proof. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and suppose Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. We will show that Duplicator wins $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ by playing in the same way as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$.

In the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, suppose from the initial round, Spoiler moves the pebble along $p_0 a_1 p_1 a_2 \ldots$. We look at what Duplicator would do in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ if Spoiler does the same move. If Duplicator skips her turn and only moves the pebble for the first time in some round $m > 0$ along a run r then in the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator also skips her turn and only moves the pebble in round m along r. She then repeats this procedure indefinitely.

Now let $v_0 v'_1 v_1 v'_2 \ldots$ and $u_0 u'_1 u_1 u'_2 \ldots$ be the plays that are obtained in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. Since in the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator moves her pebble in the same way as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, we have

$$v_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{S}), \tag{3.24}$$

$$u_i = (p_i, \overline{w'_i}, q_i, c'_i, \mathsf{S}), \tag{3.25}$$

for all $i > 0$. In other words, the configurations that are reached by Duplicator in every round of $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ only differ in the second and the fourth components. This similarly also holds for the configurations reached by Spoiler.

Since Duplicator wins the play $v_0 v'_1 v_1 v'_2 \ldots$, either there are only finitely many i such that $v'_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{D})$ with $p_i \in F^{\mathcal{A}}$ or there are infinitely many i such that $v_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{S})$ with

$$c_i = \top. \tag{3.26}$$

In the first case, there are also finitely many i such that $u'_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{D})$ with $p_i \in F^{\mathcal{A}}$ and hence Duplicator wins $u_0 u'_1 u_1 u'_2 \ldots$. In the second one, we will show that there are also infinitely many i such that $u_i = (p_i, \overline{w'_i}, q_i, c'_i, \mathsf{D})$ with $c'_i = \top$. We will show this by contradiction.

Suppose there are only finitely many such i. This implies that in the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, there is a round $i_0 > 0$ where started from this round, the counter never changes: we have $c \in \{1, \ldots, n, \bot, \top\}$ such that

$$c'_i = c \tag{3.27}$$

for all $i \geq i_0$. First note that $c \neq \top$ since otherwise in the next round, i.e. round $i_0 + 1$, the counter will changes to 1 and we have a contradiction to (3.27). We also have $c \neq \bot$ since (3.26) holds for infinitely many i. Duplicator sees an accepting state in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ as well as in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, infinitely often, and hence if $c = \bot$, there is a round $i_1 > i_0$ where the counter increases to $\top$ and we again have a contradiction to (3.27). Thus $c \in \{1, \ldots, n\}$.

We will show that this will also give us a contradiction. For all $j \in \{1, \ldots, n\}$, let w_j, w'_j be the contents of the j-th buffer in round i_0 of the games $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, respectively. First, note that $w'_c \neq \epsilon$ since otherwise in round $i_0 + 1$ the counter will increase from c and we again have a contradiction to (3.27). Furthermore, since $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, $\hat{\Sigma}' = (\Sigma_{\sigma(1)}, \ldots, \Sigma_{\sigma(n)})$, and σ a permutation, we have $w'_c = w_d$ for $c = \sigma(d)$. In the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, there is a round $i_1 \geq i_0$ where Duplicator pops the d-th buffer since otherwise the counter at one point never changes and we have a contradiction since (3.26) holds for infinitely many i. Hence in round i_1 of the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator pops the c-th buffer. The counter increases from c to $c + 1$ if $c < n$ or resets to $\bot$ if $c = n$. This again a contradiction to (3.27). Hence there is no such round i_0. We have infinitely many i such that $c'_i = \top$. Duplicator wins the play $u_0 u'_1 u_1 u'_2 \ldots$. She wins the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. The other direction can be shown similarly. □

Eliminating Buffers

There are some cases where we can eliminate some certain buffers without giving any players more power to win the game. For example, consider a buffered simulation game with two buffers in which the first one has more capacity than the second one, and every letter that is stored in the first buffer also gets stored in the second one. In such a case, the first buffer can be eliminated. This is intuitively because the first buffer is redundant to the second one. Whenever Spoiler reads a letter and a copy of the letter is pushed to the first buffer, one copy of the letter is also pushed to the second one. If Duplicator never violates the capacity constraint of the second buffer, since the first buffer is bigger than the second one, Duplicator also never violates the capacity constraint of the first buffer. Hence eliminating the first buffer does not give any disadvantage or advantage for Duplicator.

Let $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ be the game obtained from $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ by eliminating the i-th buffer because of redundancy to the i'-th buffer. By Theorem 3.5.1, without loss of generality, we can assume that $i = 1$ and $i' = 2$. If Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ then she also wins the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. She simply moves her pebble in the same way as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. In this case, it is clear that if during the play in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator resets the counter once, she also resets the counter at least once in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. The reason is because we now consider a game with less buffers and the order of the buffers does not change. If Duplicator resets the counter infinitely often in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, she also does so in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$.

Theorem 3.5.2. *For any two distributed alphabets $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, $\hat{\Sigma}' = (\Sigma_2, \ldots, \Sigma_n)$ and two capacity vectors $\kappa = (k_1, \ldots, k_n)$, $\kappa' = (k_2, \ldots, k_n)$ in which $k_1 \geq k_2$ and $\Sigma_1 \subseteq \Sigma_2$, we have*

$$\sqsubseteq^{\kappa}_{\hat{\Sigma}} \subseteq \sqsubseteq^{\kappa'}_{\hat{\Sigma}'} . \tag{3.28}$$

The other direction of (3.28) also holds. If Duplicator wins the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, she also wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ by moving her pebble in the same way as in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. However note if in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator resets the counter once, in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator might not do so. In the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, the counter might not increase from 1. This is because Duplicator pops a letter from Σ_2 that does not belong to Σ_1. However since $\Sigma_1 \subseteq \Sigma_2$, in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator eventually will pop the letter from Σ_1 in the first buffer, and in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, the counter increases from 1. Duplicator resets the counter infinitely often in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ if she also does so in the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$.

Theorem 3.5.3. *For any two distributed alphabets $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, $\hat{\Sigma}' = (\Sigma_2, \ldots, \Sigma_n)$ and two capacity vectors $\kappa = (k_1, \ldots, k_n)$, $\kappa' = (k_2, \ldots, k_n)$ in which $k_1 \geq k_2$ and $\Sigma_1 \subseteq \Sigma_2$, we have*

$$\sqsubseteq^{\kappa}_{\hat{\Sigma}} \supseteq \sqsubseteq^{\kappa'}_{\hat{\Sigma}'} .$$

Proof. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. Suppose Duplicator wins $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. In the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator simply moves her pebble in the same way as in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$.

Now let $v_0 v_1' v_1 v_2' \dots$ and $u_0 u_1' u_1 u_2' \dots$ be the plays that are obtained in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. Since in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator moves the pebble in the same way as in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, we also have the properties as in (3.24) and (3.25) for all $i > 0$.

We will show that if there are infinitely many i such that $u_i = (p_i, \overline{w_i'}, q_i, c_i', \mathsf{D})$ with

$$c_i' = \top, \tag{3.29}$$

there are also infinitely many i such that $v_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{D})$ with $c_i = \top$. We will show this by contradiction.

Suppose there are only finitely many such i. With the same reasoning as in the proof of Theorem 3.5.1, this implies that in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, there is a round $i_0 \geq 0$ where started from this round, the counter never changes from $c \in \{1, \dots, n\}$, i.e.

$$c_i = c \tag{3.30}$$

for all $i \geq i_0$.

For all $j \in \{1, \dots, n\}$, let w_j and w_j' be the content of the j-th buffer in round i_0 of the games $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, respectively. We have $w_c \neq \epsilon$ since otherwise in round $i_0 + 1$ the counter will increase from c and we have a contradiction to (3.30). Furthermore, since $\hat{\Sigma}' = (\Sigma_2, \dots, \Sigma_n)$, $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$, we have $w_c = w_{c-1}'$ for $c > 1$.

Now suppose $c > 1$. In the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, there is a round $i_1 \geq i_0$ where Duplicator pops the $(c-1)$-th buffer since otherwise, at one point the counter in $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ never changes and this contradicts that (3.29) holds for infinitely many i. Hence in round $i_1 + 1$, the counter increases from c to $c + 1$ if $c < n$ or resets to $\perp$ if $c = n$. This is again a contradiction to (3.30).

In the case where $c = 1$, since $\Sigma_1 \subseteq \Sigma_2$, we have $w_c = \pi_1(w_c')$ where π_1 is the projection to Σ_1. In the game $\mathcal{G}^{\kappa'}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, there is a round $i_2 \geq i_0$ where Duplicator pops the first letter of $\pi_1(w_c')$ from the first buffer. Otherwise, at one point the counter never changes and this again contradicts that (3.29) holds for infinitely many i. Hence in round i_2 of $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, Duplicator pops the first buffer and the counter increases from 1 which is again a contradiction to (3.30).

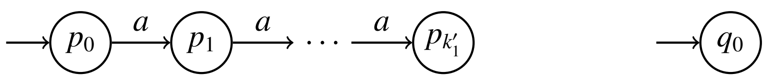
Hence there is no such round i_0. We have infinitely many i such that $c_i = \top$. Since Duplicator wins $u_0 u_1' u_2 u_2' \dots$, she also wins the play $v_0 v_1' v_1 v_2' \dots$. Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. □

Hierarchy over Capacity Vectors

As in the case of one buffer, buffered simulation with multiple buffers also gets more expressive as the capacity of the buffers grows. We have a similar hierarchy as in Theorem 3.1.4. First, let us consider an order over the capacity vectors by considering the total order on $\mathbb{N} \cup \{\omega\}$ that is extended to the point-wise comparisons on n-tuples of elements from $\mathbb{N} \cup \{\omega\}$. For any two capacity vectors $\kappa = (k_1, \dots, k_n)$ and $\kappa' = (k_1', \dots, k_n')$, we have $\kappa \leq \kappa'$ iff $k_i \leq k_i'$ for all $i \in \{1, \dots, n\}$. Furthermore, we write $\kappa < \kappa'$ iff $\kappa \leq \kappa'$ and there is $i \in \{1, \dots, n\}$ such that $k_i < k_i'$. For example, we have $(78, 6, 19) < (\omega, 6, 23)$ but $(3, 4) \nleq (5, 2)$. By considering such an order, as the capacity vector grows, buffered simulation gets more expressive. This, however, only holds with respect to a fixed distributed alphabet.

Theorem 3.5.4. *Let $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ be a distributed alphabet. For any two capacity vectos κ, κ' in which $\kappa < \kappa'$, we have $\sqsubseteq^{\kappa}_{\hat{\Sigma}} \subsetneq \sqsubseteq^{\kappa'}_{\hat{\Sigma}}$.*

Proof. For any two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma}$ and two capacity vectors $\kappa < \kappa'$, any winning strategy in $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ is also a winning strategy in $\mathcal{G}^{\kappa'}(\mathcal{A}, \mathcal{B})$. Hence for all $\kappa < \kappa'$, we have $\sqsubseteq^{\kappa}_{\hat{\Sigma}} \subseteq \sqsubseteq^{\kappa'}_{\hat{\Sigma}}$. For the strictness part, suppose $\kappa = (k_1, \ldots, k_n)$, $\kappa' = (k'_1, \ldots, k_n{}')$, and $\kappa < \kappa'$. Since $\kappa < \kappa'$, we have $k_1 < k'_1$. Now consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ where a is some arbitrary letter in Σ_1.



We have $\mathcal{A} \sqsubseteq^{\kappa'} \mathcal{B}$. This is because in the game $\mathcal{G}^{\kappa'}(\mathcal{A}, \mathcal{B})$, Duplicator wins by just skipping her turn. At the end of round k'_1, we reach the configuration $(p_{k'_1}, (a^{k'_1}, \epsilon, \ldots, \epsilon), q_0, \bot, \mathsf{S})$ which does not have any valid successor since the state $p_{k'_1}$ does not have a successor. Hence Spoiler gets stuck and Duplicator wins the play. However, we have $\mathcal{A} \not\sqsubseteq^{\kappa} \mathcal{B}$. Spoiler wins $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ by reading a^{k_1+1}. This is possible because $k_1 < k'_1$. Hence at round $k_1 + 1$, after Spoiler reads a, we reach the configuration $(p_{k_1}, (a^{k_1+1}, \epsilon, \ldots, \epsilon), q_0, \bot, \mathsf{D})$ which does not have any valid successor since the state q_0 does not have a successor. Hence Duplicator loses the game $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$. □

Hierarchy over Distributed Alphabets

Theorem 3.5.4 shows that for a fixed distributed alphabet, buffered simulations gets more expressive as the capacity of the buffers grows. It turns out that for a fixed capacity vector, buffered simulation also gets more expressive as the distributed alphabet gets “less” distributed. By “less” distributed, we mean each buffer gets associated to the same or a smaller set of letters.

To show this formally, consider the order on the distributed alphabets by considering the inclusion that is extended to the point-wise comparisons of n-tuples of subset of Σ. For any two distributed alphabets $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and $\hat{\Sigma}' = (\Sigma'_1, \ldots, \Sigma'_n)$ where $\Sigma_1 \cup \ldots \cup \Sigma_n = \Sigma'_1 \cup \ldots \cup \Sigma'_n$, we say $\hat{\Sigma}'$ is *finer* than $\hat{\Sigma}$ and write $\hat{\Sigma}' \leq \hat{\Sigma}$ if $\Sigma'_i \subseteq \Sigma_i$ for all $i \in \{1, \ldots, n\}$. The finer the distributed alphabet, the stronger is Duplicator to win the game. Intuitively, this is because in the game with a finer distributed alphabet, each letter that is read by Spoiler now is put to the same set of buffers or less. Hence Duplicator’s moves are less restricted.

If Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ then she also wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ where $\hat{\Sigma}' \leq \hat{\Sigma}$ by moving her pebble as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. Note that if Duplicator resets the counter in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ infinitely often, she will also do so in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. Since otherwise, there is a non-empty buffer, let us assume the i-th one, in $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ that is ignored by Duplicator. Since every letter that gets stored to the i-the buffer in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$ also gets stored to the i-th buffer in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, this means Duplicator also ignores the i-th buffer in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and contradicts our initial assumption. We formally show this as follows.

Theorem 3.5.5. *Let $\kappa = (k_1, \ldots, k_n)$ be a capacity vector. For any two n-tuples of distributed alphabets $\hat{\Sigma}, \hat{\Sigma}'$ in which $\hat{\Sigma} < \hat{\Sigma}'$, we have $\sqsubseteq^{\kappa}_{\hat{\Sigma}'} \subseteq \sqsubseteq^{\kappa}_{\hat{\Sigma}}$.*

Proof. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and suppose Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$. Let $\hat{\Sigma}' = (\Sigma'_1, \ldots, \Sigma'_n)$ be some distributed alphabet in which $\hat{\Sigma}' \leq \hat{\Sigma}$ and π, π' be the projections with respect to $\hat{\Sigma}$, $\hat{\Sigma}'$, respectively. In the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator moves her pebble in the same way as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$.

Now let $v_0 v_1' v_1 v_2' \dots$ and $u_0 u_1' u_1 u_2' \dots$ be the plays that are obtained in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ and in $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. Since in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator plays as in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, we also have the same properties as in (3.24) and (3.25).

We will show that if there are infinitely many round i such that $v_i = (p_i, \overline{w_i}, q_i, c_i, \mathsf{D})$ with

$$c_i = \top, \tag{3.31}$$

there are also infinitely many round i such that $u_i = (p_i, \overline{w_i}, q_i, c_i', \mathsf{D})$ with $c_i' = \top$. We will show this by contradiction.

Suppose there are only finitely many such i. By the same reasoning as in Theorem 3.5.1, this implies that in the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, there is a round $i_0 \geq 0$ where started from this round, the counter never changes from $c \in \{1, \dots, n\}$, i.e.

$$c_i' = c \tag{3.32}$$

for all $i \geq i_0$.

For all $j \in \{1, \dots, n\}$, let w_j, w_j' be the contents of the j-th buffer in round i_0 in the games $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, respectively. Since $\hat{\Sigma}' \leq \hat{\Sigma}$, we have $\pi_j'(w_j) = w_j'$ for all $j \in \{1, \dots, n\}$. Note that we have $w_c' \neq \epsilon$ since otherwise in round $i_0 + 1$, the counter will increase from c and this contradicts (3.32) holds for all $i \geq i_0$. In the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, there is some round $i_1 \geq i_0$ where Duplicator pops the first letter of $\pi_c'(w_c')$ from buffer c. Otherwise, the counter in $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$ at one point, never changes and this contradicts that (3.31) holds for infinitely many i. Hence in round i_1 of $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$, Duplicator also pops a letter from buffer c. In round $i_1 + 1$, the counter will increase to $c + 1$ if $c < n$ or resets to $\perp$ if $c = n$. This again contradicts that (3.32) holds for all $i \geq i_0$. Hence there is no such round i_0. We have infinitely many i such that $c_i' = \top$.

Since Duplicator wins $\mathcal{G}^{\kappa}_{\hat{\Sigma}}(\mathcal{A}, \mathcal{B})$, she wins $v_0 v_1' v_1 \dots$. Thus she also wins the play $u_0 u_1' u_1 \dots$. Duplicator wins the game $\mathcal{G}^{\kappa}_{\hat{\Sigma}'}(\mathcal{A}, \mathcal{B})$. □

The comparison between buffered simulation games with different distributed alphabet, however, is not particularly meaningful. Buffered simulation games are defined with respect to a distributed alphabet, and hence each distributed alphabet defines its own class of structures and the corresponding buffered simulations become relations on different classes of structures. We will see in the following chapter that the complexity and decidability issues of buffered simulation games do not depend on how the letters are distributed among the buffers, but only on how many buffers are involved and how big they are.

Chapter 4

Decidability and Complexity

In the previous chapter, we have seen the formal definition of buffered simulation. It extends the game framework of the standard fair simulation with one or multiple buffers such that Duplicator can store the letter that is read by Spoiler temporarily to the buffers before she executes it in her structure. Duplicator can have a preview of Spoiler's move and more chances to mimic it correctly. In general, each of the buffers might have a bounded or an unbounded capacity.

The definition of buffered simulation is quite simple and natural, but it is not clear whether such a simulation is still decidable. If all buffers are bounded, intuitively buffered simulation should still be decidable since the number of configurations is finite. However if some buffers are unbounded, the number of configurations is no longer finite. It is already infinite even in the case where we only have one unbounded buffer. Hence in the case where some buffers are unbounded, buffered simulation might be no longer decidable.

We study the decidability and complexity of buffered simulation in various cases. In general, we will consider the following decision problem.

$$\begin{aligned}&\textbf{Given :}\ \text{Two NBA } \mathcal{A}, \mathcal{B} \text{ over } \hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n) \text{ and a capacity vector}\\ &\qquad\quad \kappa = (k_1, \dots, k_n) \in (\mathbb{N} \cup \{\omega\})^n \qquad\qquad (4.1)\\ &\textbf{Question :}\ \text{Does Duplicator win the game } \mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})?\end{aligned}$$

Before we show the decidability and complexity of this problem, we will consider some of its simpler instances in the first three sections of this chapter. In the first one, we will consider the problem of deciding buffered simulation in which only one buffer is involved and the capacity is bounded, i.e. the instance of (4.1) in which $n = 1$ and $k_1 \in \mathbb{N}$. In the second one, we will consider a more general problem, namely deciding buffered simulation where multiple buffers are involved and all of them have a bounded capacity, i.e. the instance of (4.1) where $n \geq 1$ and $k_1, \dots, k_n \in \mathbb{N}$. In the third section, we will consider the problem of deciding buffered simulation where only one buffer is involved but the capacity is unbounded, i.e. the instance of (4.1) where $n = 1$ and $k_1 = \omega$. All of these three instances turn out to be decidable. The decidability of the first two is expected since the number of configurations in the game $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ is finite. However the decidability of the third one is quite surprising. In such a case, the number of configurations in the buffered simulation game is infinite and questions about systems with an unbounded FIFO buffer are often undecidable. For instance in [CF05], it is shown that the linear-time properties of a system with two machines and one buffer are undecidable. Nevertheless, we will see that by adding one more buffer, even with a minimal capacity, buffered simulation is no longer decidable. We will end this chapter with some sections that show the high undecidability of the problem stated in (4.1).

4.1 Simulation with One Bounded Buffer

Let us start with a simple case of buffered simulation where there is only one bounded buffer. Consider the following problem.

Given : Two NBA $\mathcal{A}, \mathcal{B}$ over Σ and $k \in \mathbb{N}$

Question : Does Duplicator win the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$?

In Theorem 3.1.8, it is shown that we can reduce any buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ where $k \in \mathbb{N} \cup \{\omega\}$ to a parity game $\mathcal{G}$ where player 1 corresponds to Spoiler and player 0 to Duplicator. The parity game $\mathcal{G}$ is played in the configuration graph of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ with a parity function that mimics the winning condition of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. The size of $\mathcal{G}$ is the same as the size of the configuration graph of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. For a bounded capacity $k \in \mathbb{N}$, we can refine Theorem 3.1.8 to the following corollary.

Corollary 4.1.1. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and a capacity $k \in \mathbb{N}$, we can construct in polynomial time a parity game $\mathcal{G}$ with priorities* 0, 1, 2 *that is played on a graph where the numbers of nodes and edges are*

$$|V| = \mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|^{k+2}),$$
$$|E| = \mathcal{O}(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{2k+3})$$

such that Duplicator *wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

Proof. Consider the parity game $\mathcal{G}$ as described in the proof of Theorem 3.1.8. The nodes in the parity game $\mathcal{G}$ are the configurations of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. By Definition 3.1.1, the set of configurations in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ is

$$V = Q^{\mathcal{A}} \times \Sigma^{\leq k+1} \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{S}, \mathsf{D}\}.$$

Since $|\Sigma^{\leq k+1}| = |\Sigma|^0 + |\Sigma|^1 + \ldots + |\Sigma|^{k+1} = \mathcal{O}(|\Sigma|^{k+2})$, the number of nodes in $\mathcal{G}$ is $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|^{k+2})$.

The edges in the parity game $\mathcal{G}$ are also the same as the edges in the configuration graph of $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. Recall also from Definition 3.1.1 that the outgoing edges from Spoiler's configurations are of the form

$$(p, w, q, b, \mathsf{S}) \to (p', wa, q, b, \mathsf{D})$$

where $w \in \Sigma^{\leq k}$ and $a \in \Sigma$. Since $|\Sigma^{\leq k}| = \mathcal{O}(|\Sigma|^{k+1})$, the number of outgoing edges from Spoiler's configuration is $\mathcal{O}\ (|\mathcal{A}|^2 \cdot |\mathcal{B}| \cdot |\Sigma|^{k+2})$. The outgoing edges from Duplicator's configuration, however, are of the form

$$(p, w, q, b, \mathsf{D}) \to (p, w', q', b', \mathsf{S})$$

where $w \in \Sigma^{\leq k+1}$ and $w' \in \Sigma^{\leq k}$. Thus the number of outgoing edges from Duplicator's configuration is $\mathcal{O}\ (|\mathcal{A}| \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{2k+3})$. Hence we have the desired result. □

Now recall that by Proposition 2.4.2 from [Jur00], deciding whether player 0 wins a parity game with priorities 0, 1, and 2 on a graph of n nodes and m edges is in time $\mathcal{O}(nm)$. Hence we have the following theorem.

Theorem 4.1.2. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and a capacity $k \in \mathbb{N}$, deciding $\mathcal{A} \sqsubseteq^k \mathcal{B}$ is in time $\mathcal{O}(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{3k+5})$.*

The Flushing Variant

In the case where we consider the flushing variant, solving buffered simulation can be done slightly easier than this. First note that the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ can also be reduced to a parity game in the same way as we reduce $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$. We can reduce it to a parity game $\mathcal{G}' = (G', c'_0, \Omega)$ where G' and c'_0 are the configuration graph and the initial configuration of the game $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, and Ω is the priority function as in (3.6).

Let $\mathcal{G}$ and $\mathcal{G}'$ be the parity games for $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, respectively. The numbers of nodes in the parity game $\mathcal{G}$ and $\mathcal{G}'$ are the same since the configurations in $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ do not differ. The number of edges in $\mathcal{G}'$, however, is slightly less than the one in $\mathcal{G}$. This is because the move of DUPLICATOR in $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is more restricted. In the configuration graph of the flushing variant $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, there are two kinds of outgoing edges from DUPLICATOR's configuration: the one that corresponds to the case where DUPLICATOR flushes the buffer and the one where she skips her turn. They are respectively of the form

$$(p, w, q, c, \mathsf{D}) \to (p, \epsilon, q', c', \mathsf{S}) \text{ and}$$
$$(p, w', q, c, \mathsf{D}) \to (p, w', q, \bot, \mathsf{S}).$$

where $p \in Q^{\mathcal{A}}$, $q, q' \in Q^{\mathcal{B}}$, $w \in \Sigma^{\leq k+1}$, $w' \in \Sigma^{\leq k}$, and $c, c' \in \{\bot, \top\}$. Since $|\Sigma^{\leq k+1}| = O(|\Sigma|^{k+2})$, the number of outgoing edges from DUPLICATOR's configuration in the configuration graph of $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is $O(|\mathcal{A}| \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{k+2})$. The numbers of outgoing edges from SPOILER's configuration in $\mathcal{G}$ and in $\mathcal{G}'$, however, are the same. It is $O(|\mathcal{A}|^2 \cdot |\mathcal{B}| \cdot |\Sigma|^{k+2})$. There is no different in how SPOILER move in the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ or $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$. Hence we have the following theorem.

Theorem 4.1.3. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and $k \in \mathbb{N}$, we can construct in polynomial time a parity game $\mathcal{G}'$ with priorities 0, 1, and 2 that is played on a graph where the numbers of nodes and edges are*

$$|V| = O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma|^{k+2}),$$
$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma|^{k+2})$$

such that DUPLICATOR *wins $\mathcal{G}^k_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ iff player 0 wins $\mathcal{G}'$.*

Again, since deciding the winner of a parity game with priorities 0, 1, and 2 is linear in the products of the numbers of nodes and edges, we have the following corollary.

Corollary 4.1.4. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and $k \in \mathbb{N}$, deciding whether $\mathcal{A} \sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$ is in time $O(|\mathcal{A}|^3\, |\mathcal{B}|^3 \cdot |\Sigma|^{2k+4})$.*

For the full-flushing variant, recall from Section 3.3 that the game can be reduced to the static $(k + 1)$-letter simulation game. Hence by Proposition 2.6.8, we have the following corollary.

Corollary 4.1.5. *For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ and a capacity $k \in \mathbb{N}$, deciding whether $\mathcal{A} \sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$ is in time and $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{2k+2})$.*

Buffered simulation with one bounded buffer together with its flushing variants, can be solved in time polynomially in the size of the automata and exponentially in the size of the alphabet where the exponent is the size of the buffer. Thus for a given fixed capacity, we can solve buffered simulation with one buffer and its flushing variant in polynomial time.

Corollary 4.1.6. *For a fixed capacity $k \in \mathbb{N}$, deciding $\sqsubseteq^k$, $\sqsubseteq^k_{\mathsf{Flush}}$, and $\sqsubseteq^k_{\mathsf{FFlush}}$ are in PTIME.*

4.2 Simulation with $n \geq 1$ Bounded Buffers

Now let us look at a more general problem. Consider the problem of deciding buffered simulation where multiple bounded buffers are involved.

> **Given :** Two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a $\kappa = (k_1, \ldots, k_n) \in \mathbb{N}^n$
> **Question :** Does Duplicator win the game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$?

As in the case of one buffer, this problem can also be reduced to a parity game. We can consider a parity game that is played on the configuration graph of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ with a priority function that mimics the winning condition of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$. The following is a generalisation of Corollary 4.1.1.

Theorem 4.2.1. *Given two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and $\kappa = (k_1, \ldots, k_n) \in \mathbb{N}^n$, we can construct in polynomial time a parity game $\mathcal{G}$ with priorities* 0, 1, *and* 2 *that is played on a graph where the numbers of nodes and edges are*

$$|V| = O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma_1|^{k_1+2} \ldots \cdot |\Sigma_n|^{k_n+2} \cdot n),$$
$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma_1|^{2k_1+3} \ldots \cdot |\Sigma_n|^{2k_n+3} \cdot n^2)$$

such that Duplicator *wins $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.*

Proof. Let $G = (V, V_S, V_D, E)$ be the configuration graph of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ and v_0 the initial configuration of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$. The parity game $\mathcal{G}$ is a triple (G, v_0, Ω) where Ω is a priority function that mimics the winning condition in $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$, i.e. for any configuration $v \in V$,

$$\Omega(v) = \begin{cases} 2 & \text{if } v = (p, \overline{w}, q, \top, \mathsf{S}) \\ 1 & \text{if } v = (p, \overline{w}, q, c, \mathsf{D}) \text{ where } p \in F^{\mathcal{A}} \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

With the same reasoning as in the proof of Theorem 3.1.8, we can show that Duplicator wins $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.

The nodes in the parity game $\mathcal{G}$ are the configurations of the game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$. Recall from Definition 3.4.2 that the set of configurations in $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ is

$$V = Q^{\mathcal{A}} \times (\Sigma_1^{\leq k_1+1} \times \ldots \times \Sigma_n^{\leq k_n+1}) \times Q^{\mathcal{B}} \times \{1, \ldots, n, \bot, \top\} \times \{\mathsf{S}, \mathsf{D}\}.$$

Since $|\Sigma_i^{\leq k_i+1}| = O(|\Sigma_i|^{k_i+2})$ for all $i \in \{1, \ldots, n\}$, the number of nodes in $\mathcal{G}$ is $O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma_1|^{k_1+2} \ldots \cdot |\Sigma_n|^{k_n+2} \cdot n)$.

The edges in the parity game $\mathcal{G}$ are also the same as the edges in the configuration graph of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$. By Definition 3.4.2, the outgoing edges from Spoiler's configuration are of the form

$$(p, \overline{w}, q, c, \mathsf{S}) \to (p', \overline{w}', q, c, \mathsf{D}),$$

where

$$\overline{w} = (w_1, \ldots, w_n),$$
$$\overline{w}' = (w_1\pi_1(a_1), \ldots, w_n\pi_n(a_n)),$$

and $w_i \in \Sigma_i^{\leq k_i}$, $\pi_i(a_i) = \epsilon$ or $a_i \in \Sigma_i$, for all $i \in \{1, \ldots, n\}$. Since $|\Sigma_i^{\leq k_i}| = O(|\Sigma_i|^{k_i+1})$ for all $i \in \{1, \ldots, n\}$, the number of outgoing edges from Spoiler's configuration is $O\ (|\mathcal{A}|^2 \cdot |\mathcal{B}| \cdot |\Sigma_1|^{k_1+2} \ldots \cdot |\Sigma_n|^{k_n+2} \cdot n)$.

The outgoing edges from DUPLICATOR's configuration, however, are of the form

$$(p, \overline{w}, q, c, \mathsf{D}) \rightarrow (p, \overline{w}', q', c', \mathsf{S}),$$

where

$$\overline{w} = (w_1, \dots, w_n),$$
$$\overline{w} = (w'_1, \dots, w'_n),$$

and $w_i \in \Sigma_i^{\leq k_i+1}$, $w'_i \in \Sigma_i^{\leq k_i}$ for all $i \in \{1, \dots, n\}$. Hence the number of outgoing edges from DUPLICATOR's configuration is $O\,(|\mathcal{A}| \cdot |\mathcal{B}|^2 \cdot |\Sigma_1|^{2k_1+3} \dots \cdot |\Sigma_n|^{2k_n+3} \cdot n^2)$. Thus the number of edges in $\mathcal{G}$ is $O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma_1|^{2k_1+3} \dots \cdot |\Sigma_n|^{2k_n+3} \cdot n^2)$. □

Since deciding the winner of a parity game with priorities 0, 1, and 2 over a graph with n nodes and m edges is in time $O(nm)$, we have the following theorem.

Theorem 4.2.2. *For any two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ and $\kappa = (k_1, \dots, k_n) \in \mathbb{N}^n$, deciding $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ is in time $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma_1|^{3k_1+5} \dots \cdot |\Sigma_n|^{3k_n+5} \cdot n^3)$.*

In the case of the flushing variant, i.e. deciding $\sqsubseteq^{\kappa}_{\mathsf{Flush}}$, the complexity is slightly better. We can show this in a similar way as in the case of one buffer. The reason is because the number of edges in the configuration graph of $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is slightly less than the one of $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$. In the configuration graph of $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, the outgoing edges from DUPLICATOR's configurations that correspond to the case where DUPLICATOR flushes the buffer and where she skips her turn are respectively of the form

$$(p, \overline{w}, q, c, \mathsf{D}) \rightarrow (p, \overline{\epsilon}, q', c', \mathsf{S}) \text{ and}$$
$$(p, \overline{w}', q, c, \mathsf{D}) \rightarrow (p, \overline{w}', q, \bot, \mathsf{S}),$$

where $p \in Q^{\mathcal{A}}$, $q, q' \in Q^{\mathcal{B}}$, $c, c' \in \{1, \dots, n, \bot, \top\}$,

$$\overline{w} = (w_1, \dots, w_n),$$
$$\overline{w}' = (w'_1, \dots, w'_n),$$
$$\overline{\epsilon} = (\epsilon, \dots, \epsilon),$$

and $w_i \in \Sigma_i^{\leq k_i+1}$, $w'_i \in \Sigma_i^{\leq k_i}$ for all $i \in \{1, \dots, n\}$. Since $|\Sigma_i^{\leq k_i+1}| = O(|\Sigma_i|^{k_i+2})$ for all $i \in \{1, \dots, n\}$, the number of outgoing edges from DUPLICATOR's configuration in the configuration graph of $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is $O(|\mathcal{A}| \cdot |\mathcal{B}|^2 \cdot |\Sigma_1|^{k_1+2} \dots \cdot |\Sigma_n|^{k_n+2} \cdot n^2)$. The numbers of outgoing edges from SPOILER's configuration in $\mathcal{G}$ and $\mathcal{G}'$ are the same. They are $O(|\mathcal{A}|^2 \cdot |\mathcal{B}| \cdot |\Sigma_1|^{k_1+2} \dots \cdot |\Sigma_n|^{k_n+2} \cdot n)$. Thus we have the following theorem.

Theorem 4.2.3. *For any two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ and $\kappa = (k_1, \dots, k_n) \in \mathbb{N}^n$, we can construct in polynomial time a parity game $\mathcal{G}'$ with priorities 0, 1, and 2 that is played on a graph where the numbers of nodes and edges are*

$$|V| = O(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |\Sigma_1|^{k_1+2} \dots \cdot |\Sigma_n|^{k_n+2} \cdot n),$$
$$|E| = O(|\mathcal{A}|^2 \cdot |\mathcal{B}|^2 \cdot |\Sigma_1|^{k_1+2} \dots \cdot |\Sigma_n|^{k_n+2} \cdot n^2)$$

such that DUPLICATOR *wins $\mathcal{G}^{\kappa}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ iff player 0 wins $\mathcal{G}'$.*

Again, by Proposition 2.4.2, the winner of a parity game with priorities 0, 1, and 2 is linear in the products of the numbers of nodes and edges. Hence we have the following corollary.

Corollary 4.2.4. *For any two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and $\kappa = (k_1, \ldots, k_n) \in \mathbb{N}^n$, deciding whether $\mathcal{A} \sqsubseteq^{\kappa}_{\mathsf{Flush}} \mathcal{B}$ is in time $O(|\mathcal{A}|^3\ |\mathcal{B}|^3 \cdot |\Sigma_1|^{2k_1+4} \ldots \cdot |\Sigma_n|^{2k_n+4} \cdot n^3)$.*

Both of the flushing variant and the general case of buffered simulation with multiple bounded buffers can be solved in time polynomially in the size of the automata and in the number of the buffers. They are however exponential in the size of the alphabets where the exponent is the capacity of the buffers. Thus for a fixed capacity vector κ, buffered simulation with multiple bounded buffers and its flushing variant can be solved in polynomial time.

Corollary 4.2.5. *For a fixed $\kappa \in \mathbb{N}^+$, deciding $\sqsubseteq^{\kappa}$ and $\sqsubseteq^{\kappa}_{\mathsf{Flush}}$ are in PTIME.*

4.3 Simulation with One Unbounded Buffer

Now let us consider a more general case where some unbounded buffer is involved. First consider the case where there is only one buffer. We will start with the flushing variant, i.e.

$$\begin{array}{l}\textbf{Given: } \text{Two NBA } \mathcal{A}, \mathcal{B} \text{ over } \Sigma \\ \textbf{Question: } \text{Does Duplicator win the game } \mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})? \end{array} \tag{4.3}$$

Note that we can also reduce the game $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ to a parity game in the same way as we reduce the game $\mathcal{G}^{k}(\mathcal{A}, \mathcal{B})$ where $k \in \mathbb{N}$. We can see it as a parity game that is played in the configuration graph of $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ with the same priority function as in (3.6). However, recall that the set of configurations in the game $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, as well as $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, is

$$V = Q^{\mathcal{A}} \times \Sigma^* \times Q^{\mathcal{B}} \times \{\top, \bot\} \times \{\mathsf{S}, \mathsf{D}\}.$$

Since the content of the buffer can be any finite word from Σ^*, there are infinitely many configurations. Reducing the game $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ to a parity game in the same way as we reduce the game $\mathcal{G}^{k}(\mathcal{A}, \mathcal{B})$, where $k \in \mathbb{N}$, will only give us a parity game on an infinite graph which does not help us in showing decidability or undecidability of (4.3).

Nevertheless, we will show that the problem in (4.3) is indeed decidable. We will first give the lower bound of solving such a problem and then its upper bound. After presenting the complexity characterisation of deciding $\sqsubseteq^{\omega}_{\mathsf{Flush}}$ completely, we will continue with a more involved problem, namely deciding $\sqsubseteq^{\omega}$.

4.3.1 The Flushing Variant

It turns out that deciding whether Duplicator wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ can no longer be done in PTIME. The problem is PSPACE-hard. We will present a reduction from the corridor tiling problem. Recall from Chapter 2 that in the corridor tiling problem, we are given a tiling system $\mathcal{T}$, a corridor width $n > 0$, and are asked whether we can tile the region $\{1, \ldots, n\} \times \mathbb{N}^+$. Such a problem can be reduced to the one of deciding $\sqsubseteq^{\omega}_{\mathsf{Flush}}$. Intuitively, we construct two NBA $\mathcal{A}$, $\mathcal{B}$ such that Spoiler's role is to produce a tiling row by row in $\mathcal{A}$ and Duplicator's role is to make sure that there is no horizontal or vertical mismatch

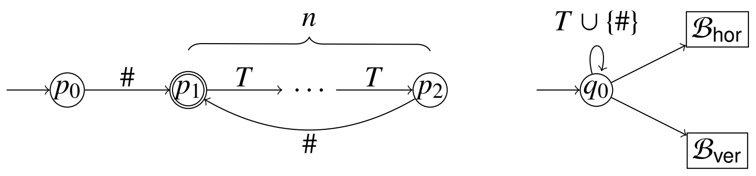


Figure 4.1: NBA $\mathcal{A}$ and $\mathcal{B}$ for the corridor tiling problem.

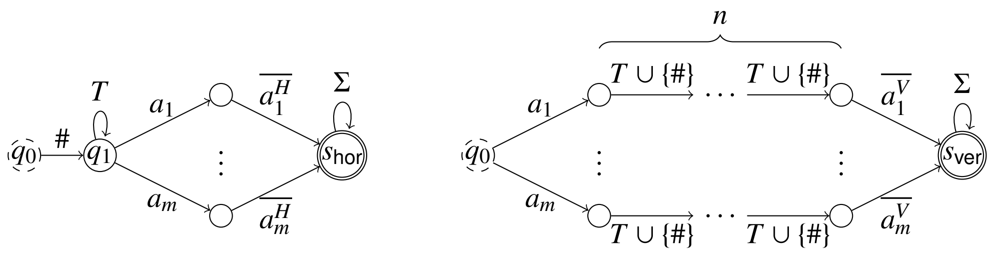


Figure 4.2: The components $\mathcal{B}_{\mathsf{hor}}$ and $\mathcal{B}_{\mathsf{ver}}$.

by playing in $\mathcal{B}$. The automaton $\mathcal{B}$ is basically equipped with two accepting sinks: s_{hor} and s_{ver} such that whenever Duplicator detects a horizontal or vertical mismatch, she simply flushes the buffer and reaches s_{hor} or s_{ver} to win the play.

Formally, given a tiling system $\mathcal{T} = (T, H, V)$ and a width $n > 0$, we construct two NBA $\mathcal{A}$, $\mathcal{B}$ as in Figure 4.1. They are defined over $\Sigma = T \cup \{\#\}$. The symbol # is used to denote the beginning of a tiling of a row. The automaton for Spoiler will produce a word of the form $\#w_1\#w_2\#\ldots$ where $w_1, w_2, \ldots \in T^n$. The words $w_1, w_2, \ldots$ correspond to the tiling of the first row, the second row, etc. The automaton for Duplicator, however, is more complex. It consists of two components: $\mathcal{B}_{\mathsf{hor}}$ that is used to detect the horizontal mismatch and $\mathcal{B}_{\mathsf{ver}}$ for the vertical mismatch. We illustrate $\mathcal{B}_{\mathsf{hor}}$ and $\mathcal{B}_{\mathsf{ver}}$ in Figure 4.2. We assume that $T = \{a_1, \ldots, a_m\}$. Moreover, for any tile $t \in T$, we denote by $\overline{t^H}$, the set of tiles that are not horizontally compatible with t, i.e. $\overline{t^H} = \{t' \in T \mid (t, t') \notin H\}$. Similarly we denote by $\overline{t^V}$, the set of tiles that are not vertically compatible with t, i.e. $\overline{t^V} = \{t' \in T \mid (t, t') \notin V\}$.

Intuitively, any infinite word $w = \#t_{1,1} \ldots t_{n,1}\#t_{1,2} \ldots t_{n,2}\# \ldots \in (\# \cdot T^n)^\omega$ is accepted by $\mathcal{B}$ if it contains a vertical or horizontal mismatch, i.e. there exist $i \in \{1, \ldots, n\}$ and $j > 0$ such that $(t_{i,j}, t_{i+1,j}) \notin H$ or $(t_{i,j}, t_{i,j+1}) \notin V$. Duplicator can read such a word w from the initial state q_0 by going to the accepting sink s_{hor} or s_{ver} and loop there for the rest of the play.

Lemma 4.3.1. *Given a tiling system $\mathcal{T} = (T, H, V)$ and a width $n > 0$, consider the two NBA $\mathcal{A}$, $\mathcal{B}$ in Figure 4.1. We have*

$$L(\mathcal{A}) = (\# \cdot T^n)^\omega \text{ and}$$
$$L(\mathcal{A}) \cap L(\mathcal{B}) = \{\#w_1\#w_2\# \ldots \in (\# \cdot T^n)^\omega \mid \exists i, j : (w_i(j), w_i(j+1)) \notin H \text{ or } w_i(j), w_{i+1}(j)) \notin V\}.$$

Now if we consider the game $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ between the two NBA $\mathcal{A}$, $\mathcal{B}$ as in Figure 4.1, Spoiler wins if there exists a corridor tiling. The strategy for Spoiler is simply to read the tiling row by row and by the construction of $\mathcal{B}$, Duplicator will never reach any accepting sinks. However, if there is no corridor tiling, Duplicator simply waits until Spoiler produces a vertical or horizontal mismatch. At that point, she flushes the buffer by going to one of the accepting sinks.

Theorem 4.3.2. *Given a tiling system $\mathcal{T} = (T, H, V)$ and a width $n > 0$, we can construct in polynomial time two NBA $\mathcal{A}, \mathcal{B}$ such that* Spoiler *wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ iff there exists a corridor tiling.*

Proof. Consider the two NBA $\mathcal{A}, \mathcal{B}$ as given in Figure 4.1. Suppose there exists a corridor tiling t. The strategy for Spoiler is to read $w = \#t_{1,1} \dots t_{n,1}\#t_{1,2} \dots t_{n,2}\# \dots$. Since for all $i \in \{1, \dots, n\}$ and $j > 0$, we have $(t_{i,j}, t_{i+1,j}) \in H$ and $(t_{i,j}, t_{i,j+1}) \in V$, by Lemma 4.3.1, $w \notin L(\mathcal{A}) \cap L(\mathcal{B})$, but $w \in L(\mathcal{A})$. Hence $w \notin L(\mathcal{B})$. There is no accepting run over w in $\mathcal{B}$. Spoiler wins because he forms an accepting run, but Duplicator does not.

For the reverse direction, suppose there is no corridor tiling. Intuitively, the strategy for Duplicator is to wait until we reach some configuration

$$(p_2, w\#t_1 \dots t_n\#t'_1 \dots t'_n, q_0, \bot, \mathsf{D}) \tag{4.4}$$

where $w \in (\# \cdot T^n)^*$, $t_1, t'_1 \dots, t_n, t'_n \in T$, and there is $i \in \{1, \dots, n\}$ such that $(t_i, t_{i+1}) \notin H$ or $(t_i, t'_i) \notin V$. In the first case, Duplicator pops w by looping in q_0, pops $\#t_1 \dots t_{i-1}$ by going to q_1, and then pops $t_i t_{i+1}$ by going to the accepting sink s_{hor}. In the second one, Duplicator pops $w\#t_1 \dots t_{i-1}$ by looping in q_0 and then pops $t_i \dots t_n\#t'_1 \dots t'_i$ by going to the accepting sink s_{ver}.

If we never reach a configuration as in (4.4) then Spoiler produces the word $\#t_{1,1} \dots t_{n,1}$ $\#t_{1,2} \dots t_{n,2}\# \dots$ where for all $i \in \{1, \dots, n\}$ and $j > 0$, $(t_{i,j}, t_{i+1,j}) \in H$ and $(t_{i,j}, t_{i,j+1}) \in V$. In such a case, we can construct a corridor tiling $t : \{1, \dots, n\} \times \mathbb{N}^+ \to T$ where $t(i, j) = t_{i,j}$ which contradicts our initial assumption. Hence the play eventually reaches a configuration as in (4.4) and Duplicator will reach the accepting sink s_{hor} or s_{ver}. From the accepting sink, it is not hard to see that Duplicator can play accordingly and win the play. □

Note that the size of the automata $\mathcal{A}$, $\mathcal{B}$ in Figure 4.1 is polynomially larger than the given tiling system and corridor width. We also have a polynomial-time reduction from the corridor tiling problem to the flushing variant of buffered simulation with one unbounded buffer. Since by Proposition 2.2.4, solving the corridor tiling problem is PSPACE-hard, deciding whether Duplicator wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is also PSPACE-hard. Hence we have the following corollary.

Corollary 4.3.3. *Deciding $\sqsubseteq^{\omega}_{\mathsf{Flush}}$ is PSPACE-hard.*

Upper Bound

This lower bound is indeed tight. We will show that deciding whether Duplicator wins the flushing variant $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ can be done in PSPACE. Before we show this, let us first denote the set of Spoiler's configurations with an empty buffer by V_ϵ, i.e. the configuration (p, w, q, b, S) is in V_ϵ iff $w = \epsilon$. Moreover, let us also denote the set of Spoiler's configurations that are winning for Spoiler by W_S, i.e. a configuration (p, w, q, b, S) is in W_S iff Spoiler has a winning strategy to win the play that is started from (p, w, q, b, S). We will give an algorithm that computes the set of Spoiler's configurations with an empty buffer that are winning for Spoiler, i.e. the set $\mathsf{WinS} = W_S \cap V_\epsilon$. Once we have WinS, we can easily decide whether Spoiler wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ by checking whether the initial configuration is in such a set.

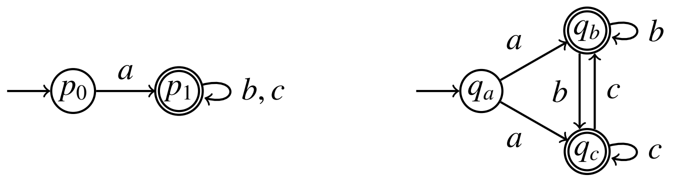
Fixpoint Iteration for Computing WinS

The set WinS can be computed via fixpoint iteration. For the base case, we consider the set of configurations $\mathsf{WinS}^0 \subseteq V_\epsilon$ in which SPOILER wins by forming an accepting run that cannot be mimicked by DUPLICATOR. First, for any automaton $\mathcal{A}$ and a state p, let us denote with $\mathcal{A}_p$, the automaton $\mathcal{A}$ where p is its initial state, i.e. $\mathcal{A}_p = (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, E^{\mathcal{A}}, p, F^{\mathcal{A}})$. We define the set WinS^0 as follows.

Definition 4.3.4. For any two NBA $\mathcal{A}$, $\mathcal{B}$,

$$\mathsf{WinS}^0 = \{(p, \epsilon, q, b, \mathsf{S}) \in V_\epsilon \mid L(\mathcal{A}_p) \not\subseteq L(\mathcal{B}_q)\}. \tag{4.5}$$

Example 4.3.5. Consider again two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 3.2.1. For convenience, we present them again as follows.



The initial configuration $(p_0, \epsilon, q_a, \bot, \mathsf{S})$ does not belong to WinS^0 since $L(\mathcal{A}) = L(\mathcal{B})$, and hence $L(\mathcal{A}_{p_0}) \subseteq L(\mathcal{B}_{q_a})$. The configurations $(p_1, \epsilon, q_b, \top, \mathsf{S})$ and $(p_1, \epsilon, q_c, \top, \mathsf{S})$, however, are in WinS^0. Note that

$$\begin{aligned} L(\mathcal{A}_{p_1}) &= (b \cup c)^\omega, \\ L(\mathcal{B}_{q_b}) &= b \cdot (b \cup c)^\omega, \text{ and} \\ L(\mathcal{B}_{q_c}) &= c \cdot (b \cup c)^\omega. \end{aligned}$$

Thus $L(\mathcal{A}_{p_1}) \not\subseteq L(\mathcal{B}_{q_b})$ and $L(\mathcal{A}_{p_1}) \not\subseteq L(\mathcal{B}_{q_c})$. In fact, we have

$$\mathsf{WinS}^0 = \{(p, \epsilon, q, \bot, \mathsf{S}), (p, \epsilon, q, \top, \mathsf{S}) \in V_\epsilon \mid (p, q) \in Q^{\mathcal{A}} \times Q^{\mathcal{B}} \setminus \{(p_0, q_a)\}\}.$$

From any configuration $(p, \epsilon, q, \bot, \mathsf{S})$ or $(p, \epsilon, q, \top, \mathsf{S})$ in WinS^0, SPOILER simply considers the word $w \in L(\mathcal{A}_p) \setminus L(\mathcal{B}_q)$ and then plays the accepting run of w in $\mathcal{A}$. DUPLICATOR would never form a corresponding accepting run and loses the play.

Proposition 4.3.6. $\mathsf{WinS}^0 \subseteq \mathsf{WinS}$.

Proof. Let $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}^0$. By definition of WinS^0, there is a word $w \in L(\mathcal{A}_p)$ such that $w \notin L(\mathcal{B}_q)$. Let ρ be an accepting run over w in $\mathcal{A}_p$. The winning strategy for SPOILER from the configuration $(p, \epsilon, q, b, \mathsf{S})$ in $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ is to play ρ. Since $w \notin L(\mathcal{B}_q)$, there is no accepting run over w from q. Hence SPOILER forms an accepting run, but DUPLICATOR does not. We obtain a play $v_0 v_1 \ldots$ such that there are infinitely many i such that $v_i = (p, w, q, b, \mathsf{D})$ with $p \in F^{\mathcal{A}}$ and finitely many i such that $v_i = (p, w, q, \top, \mathsf{S})$. SPOILER wins the play. □

The set WinS indeed is the least set that contains WinS^0 such that from any configuration in WinS, SPOILER can force DUPLICATOR to eventually proceed to WinS^0. We will show this by considering the set of attractors. For any set of configurations $U \subseteq V_\epsilon$, the set $\mathsf{Attr}_S(U)$ is defined as the set of configurations in V_ϵ from which SPOILER can play a certain accepting run and force DUPLICATOR to eventually flush the buffer and proceed to U. We formally denote this as follows.

Definition 4.3.7. Let $\mathcal{A}$, $\mathcal{B}$ be two NBA. For any $U \subseteq V_\epsilon$, the set of *attractors* of U is

$$\begin{aligned}\mathsf{Attr}_S(U) = \{(p,\epsilon,q,b,\mathsf{S}) \in V_\epsilon \mid \exists \rho = pa_1p_1a_2p_2\ldots \in \mathsf{AccRun}(\mathcal{A}_p)& \\ \text{such that } \forall \rho' = qa_1q_1a_2q_2\ldots \in \mathsf{AccRun}(\mathcal{B}_q),\ \text{there}& \\ \text{are finitely many } i \text{ with } (p_i,\epsilon,q_i,\top,\mathsf{S}) \notin U\}.&\end{aligned}$$

Example 4.3.8. Consider again the two NBA $\mathcal{A}$, $\mathcal{B}$ in Example 4.3.5 and the set $U = \{(p_1,\epsilon,q_b,\top,\mathsf{S}),(p_1,\epsilon,q_c,\top,\mathsf{S})\}$. The initial configuration $(p_0,\epsilon,q_a,\bot,\mathsf{S})$ is in the set $\mathsf{Attr}_S(U)$ since we can consider the accepting run $\rho = p_0a(p_1bp_1c)^\omega$ over the word $a(bc)^\omega$. There is only one accepting run in $\mathcal{B}$ over the same word, i.e. $\rho' = q_aa(q_bbq_cc)^\omega$, and we have $(p_1,\epsilon,q_b,\top,\mathsf{S}),(p_1,\epsilon,q_c,\top,\mathsf{S}) \in U$. Hence by definition, $(p_0,\epsilon,q_a,\bot,\mathsf{S}) \in \mathsf{Attr}_S(U)$.

The fixpoint-algorithm starts from WinS^0. In every step, we add the attractors of the current set until we reach a fixed point. We eventually reach a fixed point since we only add more configurations and there are only finitely many configurations that we can add. Let us denote such a fixed point with W.

Definition 4.3.9. For all $i \in \mathbb{N}$, let WinS^0 as in (4.5),

$$\mathsf{WinS}^{i+1} = \mathsf{WinS}^i \cup \mathsf{Attr}_S(\mathsf{WinS}^i),$$

and $W = \mathsf{WinS}^{i_0}$ where $\mathsf{WinS}^{i_0} = \mathsf{WinS}^i$ for all $i > i_0$.

Intuitively, if a configuration $(p,\epsilon,q,b,\mathsf{S})$ does not belong to the fixed point W then from such a configuration, Duplicator eventually can proceed to some other configuration $(p',\epsilon,q',\top,\mathsf{S})$ that does not belong to W.

Lemma 4.3.10. *Consider a play in* $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A},\mathcal{B})$ *from some configuration* $(p,\epsilon,q,b,\mathsf{S}) \notin W$. Duplicator *has a strategy such that either* Spoiler *loses or the play eventually proceeds to some configuration*

$$(p',\epsilon,q',\top,\mathsf{S}) \notin W.$$

Proof. The strategy for Duplicator is as follows. She skips her turn until the play reaches some configuration $(p',w,q,\bot,\mathsf{D})$ where there exists q' such that

$$q \xrightarrow[F]{w} q'$$

and $(p',\epsilon,q',\top,\mathsf{S}) \notin W$. In such a case, Duplicator flushes the buffer and proceeds to the configuration $(p',\epsilon,q',\top,\mathsf{S})$.

If Spoiler forms an accepting run $\rho = pa_1p_1\ldots$ then Duplicator eventually flushes the buffer. We will show this by contradiction. Suppose Spoiler forms such an accepting run ρ, but Duplicator never flushes the buffer. Let $\rho' = qa_1q_1\ldots$ be an accepting run in $\mathcal{B}$ and $i_0 > 0$ be the smallest index such that $q_{i_0} \in F^{\mathcal{B}}$. By definition of the strategy, Duplicator does not flush the buffer in any round $i \geq i_0$ because $(p_i,\epsilon,q_i,\top,\mathsf{S}) \in W$. Hence there are only finitely many indices i such that $(p_i,\epsilon,q_i,\top,\mathsf{S}) \notin W$. By definition of attractor, $(p,\epsilon,q,b,\mathsf{S}) \in \mathsf{Attr}_S(W)$. Moreover, by definition of W, $(p,\epsilon,q,\top,\mathsf{S}) \in W$. This contradicts our initial assumption. Thus either Spoiler does not form an accepting run and in this case Duplicator wins, or Duplicator eventually flushes the buffer and proceeds to some configuration $(p',\epsilon,q',\top,\mathsf{S}) \notin W$. □

By using this lemma, we can show that Duplicator wins from any configuration $(p, \epsilon, q, b, \mathsf{S})$ that does not belong to the fixed point W. She simply flushes the buffer only if she reaches some configuration $(p', \epsilon, q', \top, \mathsf{S})$ that also does not belong to W.

Lemma 4.3.11. $\mathsf{WinS} \subseteq W$.

Proof. Let $(p, \epsilon, q, b, \mathsf{S}) \notin W$. Thus we have $(p, \epsilon, q, b, \mathsf{S}) \notin \mathsf{WinS}^i$ for all $i \in \mathbb{N}$. Duplicator wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ from such a configuration. Initially, she plays with the strategy as we have described in the proof of Lemma 4.3.10. By doing so, either she wins, or the play eventually proceeds to some configuration $(p', \epsilon, q', \top, \mathsf{S}) \notin W$. From configuration $(p', \epsilon, q', \top, \mathsf{S})$, since it does not belong to W, we can repeat the procedure again. We do this indefinitely. Hence either Duplicator eventually wins by using the strategy as described in the proof of Proposition 4.3.10 or she proceeds to some configuration of the form $(p', \epsilon, q', \top, \mathsf{S})$ infinitely often. In such a case, Duplicator also wins the play. Hence $(p, \epsilon, q, b, \mathsf{S}) \notin \mathsf{WinS}$. □

We will show that the reverse direction of this lemma also holds. If a configuration $(p, \epsilon, q, b, \mathsf{S})$ belongs to W then it also belongs to WinS^i for some $i \in \mathbb{N}$. Hence Spoiler can force the play to proceed to some configuration that belongs to WinS^{i-1}, then WinS^{i-2}, and so on, until we reach the one that belongs to WinS^0. From such a configuration, by Proposition 4.3.6, we know that Spoiler will win the play.

Theorem 4.3.12. $\mathsf{WinS} = W$.

Proof. The left-to-right direction holds by Lemma 4.3.11. For the right-to-left direction, let $(p, \epsilon, q, b, \mathsf{S}) \in W$. By Definition 4.3.9, we have an $i \in \mathbb{N}$ such that $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}^i$. If $i = 0$, by Proposition 4.3.6, $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}$. Hence we have the desired result. Now suppose $i > 0$. By Definition 4.3.9, either $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}^{i-1}$ or $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(\mathsf{WinS}^{i-1})$. In the first case, by the induction hypothesis, $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}$ and hence we also have the desired result. In the second case, by the definition of attractor, there exists an accepting run $\rho = pa_1p_1a_2\ldots$ in $\mathcal{A}$ such that for any accepting run $\rho' = qa_1q_1a_2\ldots$ in $\mathcal{B}$, there are only finitely many j such that $(p_j, \epsilon, q_j, \top, \mathsf{S}) \notin \mathsf{WinS}^{i-1}$. From $(p, \epsilon, q, b, \mathsf{S})$, the strategy for Spoiler is to play the run ρ indefinitely until Duplicator flushes the buffer and proceeds to some configuration $(p', \epsilon, q', \top, \mathsf{S}) \in \mathsf{WinS}^{i-1}$. If Duplicator never does so then either she does not form an accepting run, or she indeed forms an accepting run $\rho'_0 = qa_1q'_1a_2q'_2\ldots$ but never proceeds to such a configuration. Let $j_1, j_2, \ldots$ be the rounds where Duplicator flushes the buffer and sees some accepting state. Hence $(p_{j_1}, \epsilon, q'_{j_1}, \top, \mathsf{S}), (p_{j_2}, \epsilon, q'_{j_2}, \top, \mathsf{S}), \ldots \notin \mathsf{WinS}^{i-1}$. There are infinitely many j such that $(p_j, \epsilon, q'_j, \top, \mathsf{S}) \notin \mathsf{WinS}^{i-1}$. This contradicts $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(\mathsf{WinS}^{i-1})$. Thus if Spoiler plays ρ, either Duplicator does not form an accepting run or she eventually flushes the buffer and proceeds to some configuration $(p', \epsilon, q', \top, \mathsf{S}) \in \mathsf{WinS}^{i-1}(W)$. In the first case, Spoiler wins because he forms an accepting run. In the second case, Spoiler also wins because by the induction hypothesis, such a configuration is winning for Spoiler. We have $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{WinS}$. □

Hence the fixed point W is indeed WinS. We can compute WinS, by first computing $W_0 = \mathsf{WinS}^0$, then $W_1 = W_0 \cup \mathsf{Attr}_S(W_0)$, then $W_2 = W_1 \cup \mathsf{Attr}_S(W_1)$, etc. until we reach a fixed point.

The Complexity of Computing WinS

For any flushing variant game $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, given a set $U \subseteq V_\epsilon$ and some configuration $(p, \epsilon, q, b, \mathsf{S}) \in V_\epsilon$, it is not obvious how difficult it is to decide the membership $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$. It turns out that we can reduce such a membership problem to a language non-inclusion between two generalised Büchi automata (GNBA). Intuitively, we construct two GNBA $\mathcal{A}'$ and $\mathcal{B}'$ over the transitions of $\mathcal{A}$ such that $\mathcal{A}'$ accepts the sequence of transitions $(p, a_1, p_1)\,(p_1, a_2, p_2)\ldots$ that corresponds to the accepting run of $\mathcal{A}$, and the automaton $\mathcal{B}'$ only accepts such a sequence if there exists an accepting run $qa_1q_2a_2\ldots$ in $\mathcal{B}$ such that there are infinitely many i with $(p_i, \epsilon, q_i, \top, \mathsf{S}) \notin U$. In this way, if there is no language inclusion between $\mathcal{A}', \mathcal{B}'$, there is an accepting run $pa_1p_1a_2\ldots$ in $\mathcal{A}$ such that for all accepting run $qa_1q_2a_2\ldots$ in $\mathcal{B}$, there are only finitely many i where $(p_i, \epsilon, q_i, \top, \mathsf{S}) \in U$, and hence we have the membership $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$.

Lemma 4.3.13. *Deciding whether* $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$ *is in PSPACE.*

Proof. Suppose Spoiler's and Duplicator's automata are $\mathcal{A} = (Q^{\mathcal{A}}, \Sigma, E^{\mathcal{A}}, p_0, F^{\mathcal{A}})$ and $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, E^{\mathcal{A}}, q_0, F^{\mathcal{B}})$. Consider two GNBA

$$\begin{aligned}\mathcal{A}' &= (Q^{\mathcal{A}}, \Sigma', E^{\mathcal{A}'}, p, \{F^{\mathcal{A}}\}),\\ \mathcal{B}' &= (Q^{\mathcal{A}} \times Q^{\mathcal{B}}, \Sigma', E^{\mathcal{B}'}, (p, q), \{Q^{\mathcal{A}} \times F^{\mathcal{B}}, U'\})\end{aligned}$$

where $\Sigma' = E^{\mathcal{A}}$,

$$\begin{aligned}E^{\mathcal{A}'} &= \{r \xrightarrow{(r,a,r')} r' \mid (r, a, r') \in E^{\mathcal{A}}\},\\ E^{\mathcal{B}'} &= \{(r, s) \xrightarrow{(r,a,r')} (r', s') \mid (s, a, s') \in E^{\mathcal{B}}\},\end{aligned}$$

and

$$U' = \{(p', q') \in Q^{\mathcal{A}} \times Q^{\mathcal{B}} \mid (p', \epsilon, q', \top, \mathsf{S}) \in V_\epsilon \setminus U\}.$$

Now suppose $L(\mathcal{A}') \subseteq L(\mathcal{B}')$. Let $\rho = pa_1p_1a_2\ldots$ be an accepting run in $\mathcal{A}$. Hence the word $w = (p, a_1, p_1)\,(p_1, a_2, p_2)\ldots$ is accepted by $\mathcal{A}'$. Since $L(\mathcal{A}') \subseteq L(\mathcal{B}')$, the word w is also accepted by $\mathcal{B}'$. There is an accepting run

$$(p, q) \xrightarrow{(p,a_1,p_1)} (p_1, q_1) \xrightarrow{(p_1,a_2,p_2)} (p_1, q_1) \ldots$$

over w in $\mathcal{B}'$. By the acceptance condition of $\mathcal{B}'$, the run $qa_1q_1a_2\ldots$ is accepting in $\mathcal{B}$ and there are infinitely many i such that $(p_i, \epsilon, q_i, \top, \mathsf{S}) \notin U$. Hence by definition, $(p, \epsilon, q, b, \mathsf{S}) \notin \mathsf{Attr}_S(U)$.

For the reverse direction, suppose $L(\mathcal{A}') \not\subseteq L(\mathcal{B}')$. There exists $w = (p, a_1, p_1)\,(p_1, a_2, p_2)\ldots \in \Sigma'^{\omega}$ such that w is accepted by $\mathcal{A}'$, but not by $\mathcal{B}'$. Let

$$(p, q) \xrightarrow{(p,a_1,p_1)} (p_1, q_1) \xrightarrow{(p_1,a_2,p_2)} (p_2, q_2) \ldots$$

be an infinite run in $\mathcal{B}'$ over w. Since $w \notin L(\mathcal{B}')$, the run is not accepting in $\mathcal{B}'$. Either $qa_1q_1a_2\ldots$ is not accepting in $\mathcal{B}$ or there are only finitely many i such that $(p_i, \epsilon, q_i, \top, \mathsf{S}) \notin U$. Thus $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$.

We have $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$ iff $L(\mathcal{A}') \not\subseteq L(\mathcal{B}')$. We can reduce the problem of deciding $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$ to the problem of deciding language non inclusion $L(\mathcal{A}') \not\subseteq L(\mathcal{B}')$ between two GNBA $\mathcal{A}'$ and $\mathcal{B}'$. By Proposition 2.3.18, deciding whether $L(\mathcal{A}') \not\subseteq L(\mathcal{B}')$ is in PSPACE. Hence deciding whether $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$ is also in PSPACE. □

Algorithm 1 Deciding $\mathcal{A} \sqsubseteq^{\omega}_{\mathsf{Flush}} \mathcal{B}$.

```
 1: $W \leftarrow \mathsf{WinS}^0$
 2: $W' \leftarrow \mathsf{WinS}^0 \cup \mathsf{Attr}_S(\mathsf{WinS}^0)$
 3: while $W \neq W'$ do
 4:     $W \leftarrow W'$
 5:     $W' \leftarrow W \cup \mathsf{Attr}_S(W)$
 6: end while
 7:
 8: if $(p_0, \epsilon, q_0, \bot, \mathsf{S}) \in W$ then return no
 9: else yes
10: end if
```

By using this lemma, for any set $U \subseteq V_\epsilon$, the set $\mathsf{Attr}_S(U)$ can be computed in PSPACE. We simply check whether for every SPOILER's configuration $(p, \epsilon, q, b, \mathsf{S})$, we have $(p, \epsilon, q, b, \mathsf{S}) \in \mathsf{Attr}_S(U)$. Since there are only polynomially many such configurations, we can also compute $\mathsf{Attr}_S(U)$ in PSPACE.

Corollary 4.3.14. *The set* $\mathsf{Attr}_S(U)$ *can be computed in PSPACE.*

The set WinS^0 can also be computed in PSPACE. Note that by Definition 4.5, deciding whether some configuration $(p, \epsilon, q, b, \mathsf{S})$ is in WinS^0 can be done by checking language non-inclusion between two Büchi automata. Since there are also polynomially many such configurations, we have the following corollary.

Corollary 4.3.15. *The set* WinS^0 *can be computed in PSPACE.*

We can then decide whether SPOILER wins $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, by first computing the set WinS. Corollary 4.3.15 and Corollary 4.3.14 show that such a set can be computed in PSPACE. After we have WinS, we check the membership of the initial configuration of $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ in WinS.

Theorem 4.3.16. *Deciding whether* DUPLICATOR *wins* $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ *is in PSPACE.*

Proof. Consider the algorithm depicted in Algorithm 1. By Corollary 4.3.15 and Corollary 4.3.14, the first two steps can be done in PSPACE. Each while-loop iteration can also be done in PSPACE and there are at most $2 \cdot |\mathcal{A}| \cdot |\mathcal{B}|$ many iterations. Checking whether the initial configuration $(p_0, \epsilon, q_0, \bot, \mathsf{S})$ is in W can also be done in polynomial time since $|W| \leq 2 \cdot |\mathcal{A}| \cdot |\mathcal{B}|$. Hence Algorithm 1 runs in PSPACE. □

Together with Corollary 4.3.3, we have characterised the complexity for solving the flushing variant $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ completely.

Corollary 4.3.17. *Deciding* $\sqsubseteq^{\omega}_{\mathsf{Flush}}$ *is PSPACE-complete.*

4.3.2 Lower Bound for Deciding $\sqsubseteq^{\omega}$

Now one might ask if the problem of deciding the general case, i.e. $\sqsubseteq^{\omega}$, is also in PSPACE. Unfortunately this is not the case. We will show that for any two NBA $\mathcal{A}$, $\mathcal{B}$, the problem of deciding whether $\mathcal{A} \sqsubseteq^{\omega} \mathcal{B}$ is indeed EXPTIME-hard. We will give a reduction from the game variant of the corridor tiling problem.

Corridor Tiling Game

Recall from Section 2.2 that there is a variant of the corridor tiling problem, where beside a tiling system $\mathcal{T} = (T, H, V)$ and a corridor width $n > 0$, we are also given initial and final tiles $t_I, t_F \in T$ and then asked whether there exists a height $m \in \mathbb{N}$ and a tiling in the region $\{1, \ldots, n\} \times \{1, \ldots, m\}$ that uses t_I as the first tile and t_F in the last tile.

The game variant of this problem is played between two players: Starter (female) and Completer (male). The objective of Completer is to construct a valid tiling row by row whereas Starter tries to prevent this by choosing the first tile of each row. Starter considers the initial tile for the first row and Completer wins if eventually the final tile is used in some row. Hence in each round $i > 0$, the play proceeds as follows.

1. Starter chooses $t_{1,i} \in T$, the first tile of row i, such that $(t_{1,i-1}, t_{1,i}) \in V$. If $i = 1$ then $t_{1,i} = t_I$.
2. Completer then completes row i. She chooses $t_{2,i}, \ldots, t_{n,i} \in T$ such that
   - $(t_{1,i}, t_{2,i}), (t_{2,i}, t_{3,i}) \ldots, (t_{n-1,i}, t_{n,i}) \in H$ and
   - $(t_{1,i-1}, t_{1,i}), (t_{2,i-1}, t_{2,i}) \ldots, (t_{1,i-1,}, t_{1,i}) \in V$ if $i > 1$.

If one of the players gets stuck then the opponent wins. If the play goes on for infinitely many rounds and the final tile is never used then Starter wins. Otherwise Completer wins immediately if at some round $m > 0$, there exist j such that $t_{j,m} = t_F$. We formally define the game as follows.

Definition 4.3.18. A *corridor tiling game* is $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_I, t_F, n) = ((V, V_{\mathsf{Comp}}, V_{\mathsf{St}}, E), v_0, \mathsf{Play})$ where $V_{\mathsf{Comp}} \subseteq T^n \times \{\mathsf{St}\}$ and $V_{\mathsf{St}} \subseteq T^n \cup \{\epsilon\} \times T \times \{\mathsf{Comp}\}$ are respectively Starter and Completer's configurations. The edge relation E is defined as follows.

$$((t_1 \ldots t_n, \mathsf{St}), (t_1 \ldots t_n, r_1, \mathsf{Comp})) \in E \quad \text{iff} \quad (t_1, r_1) \in V \text{ and for all } 1 \leq i \leq n,\ t_i \neq t_F,$$

$$((t_1 \ldots t_n, r_1, \mathsf{Comp}), (r_1 \ldots r_n, \mathsf{St})) \in E \quad \text{iff} \quad \begin{array}{l} \text{for all } 1 \leq i < n,\ (r_i, r_{i+1}) \in H \text{ and} \\ \text{for all } 1 \leq i \leq n,\ (t_i, r_i) \in V. \end{array}$$

The initial configuration is $v_0 = (\epsilon, t_I, \mathsf{Comp})$ and the winning condition is Play, the set of all infinite plays in $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_I, t_F, n)$.

Note that by definition of the winning condition, Starter wins any infinite play. However, the corridor tiling game is defined such that if Completer eventually produces a row with the final tile then Starter gets stuck, and hence she loses the play immediately. Thus Completer wins iff the final tile is eventually used or Starter gets stuck because he cannot choose a tile for the first row that obeys the vertical matching relation.

Proposition 4.3.19 ([Boa97, Chl86]). *Deciding whether* Completer *wins the corridor tiling game* $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_I, t_F, n)$ *where n is encoded unarily is EXPTIME-complete.*

Reduction from the Corridor Tiling Game

We will use the corridor tiling game to show the EXPTIME-hardness of solving the buffered simulation game with one unbounded buffer. Intuitively, we can use the same principle as in the case of the flushing variant where Spoiler's role is to produce a tiling row by row and DUPLICATOR checks whether there is a horizontal or vertical mismatch. However in this case, we also need to encode the interaction between the players. SPOILER, who will mimic COMPLETER's move, has to produce a row after DUPLICATOR chooses the first tile and DUPLICATOR, who will mimic STARTER's move, has to choose the first tile of the next row after SPOILER completes the current row. The players move alternatingly. This is in contrast to the nature of buffered simulation where the players may not move alternatingly. In the game $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, DUPLICATOR can skip her turn, even for unboundedly many rounds.

Nevertheless, we will show that we can still encode the interaction in the corridor tiling game to the one between SPOILER and DUPLICATOR in the buffered simulation game. The trick is that we consider two NBA $\mathcal{A}$, $\mathcal{B}$ such that after producing the tiling of a row, SPOILER can repeat it indefinitely until DUPLICATOR announces the first tile of the next row. DUPLICATOR will do so by moving her pebble to some certain state. DUPLICATOR then mimics the repeated tiling indefinitely until SPOILER stops repeating and reading the tiling of the next row.

We illustrate the two NBA $\mathcal{A}, \mathcal{B}$ that encode such interactions in Figure 4.3. They are defined over $\Sigma = T \cup \{\#, \$\}$ where # is a special symbol that indicates the beginning of a row and \$ for the repetition of a row. Without loss of generality, we assume that there are $m > 0$ many tiles, i.e. $T = \{a_1, \ldots, a_m\}$, and a_1 is the initial tile. For any tile $t \in T$, we denote with $\bar{t}$ the set of tiles that are not t, i.e. $\bar{t} = T \backslash \{t\}$,

The automaton $\mathcal{A}$ is very simple. It accepts any word of the form $w = c_0 w_1 c_1 w_2 \ldots$ where $c_0 = \#$ and for all $i > 0$, $w_i \in T^n$ and $c_i \in \{\#, \$\}$. The automaton $\mathcal{B}$, however, is more involved. It basically consists of m main states q_{a_1}, ..., q_{a_m} and a component that starts from q_2 which can reach the accepting sink s_{id}. The state q_2 is reachable from any main state by a #-transition. Any main state q_t can reach the accepting sink s_t by reading $\#\bar{t}$. Moreover, the main states $q_{a_1}, \ldots, q_{a_m}$ are also reachable from one another by a #-transition.

To illustrate what DUPLICATOR can do in the game $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, consider a play that starts from the initial configuration $(p_0, \epsilon, q_{a_1}, \bot, \mathsf{S})$. Initially, SPOILER is forced to read # and then a word $w \in T^n$ that corresponds to the tiling of the first row where the first tile is a_1, i.e. $w(1) = a_1$. If the first tile is not a_1 then DUPLICATOR can go to the accepting sink s_{a_1} by reading $\#w(1)$ and win the play. Moreover, after reading w, SPOILER has to repeat it. She is forced to read $\$w\$w\$\ldots$ since otherwise DUPLICATOR can go to the accepting sink s_{id} or one of the accepting sinks $s_{a_1}, \ldots, s_{a_m}$. We formulate this observation in the following lemma.

Lemma 4.3.20. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ as in Figure 4.4 and a play in $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$ from some configuration*

$$(p_1, w\#, q_t, b, \mathsf{D})$$

where $w \in T^n \cup \{\epsilon\}$, $t \in T$, and $b \in \{\top, \bot\}$.

- DUPLICATOR *has a strategy such that either* SPOILER *loses or the play proceeds to some configuration*

$$(p_1, \#w', q_t, \bot, \mathsf{D}) \tag{4.6}$$

  where $w' \in T^n$ and $w'(1) = t$.

- *From (4.6), for any $t' \in T$,* DUPLICATOR *has a strategy such that either* SPOILER *loses or the play proceeds to*

$$(p_1, w'\#, q_{t'}, \top, \mathsf{D}). \tag{4.7}$$

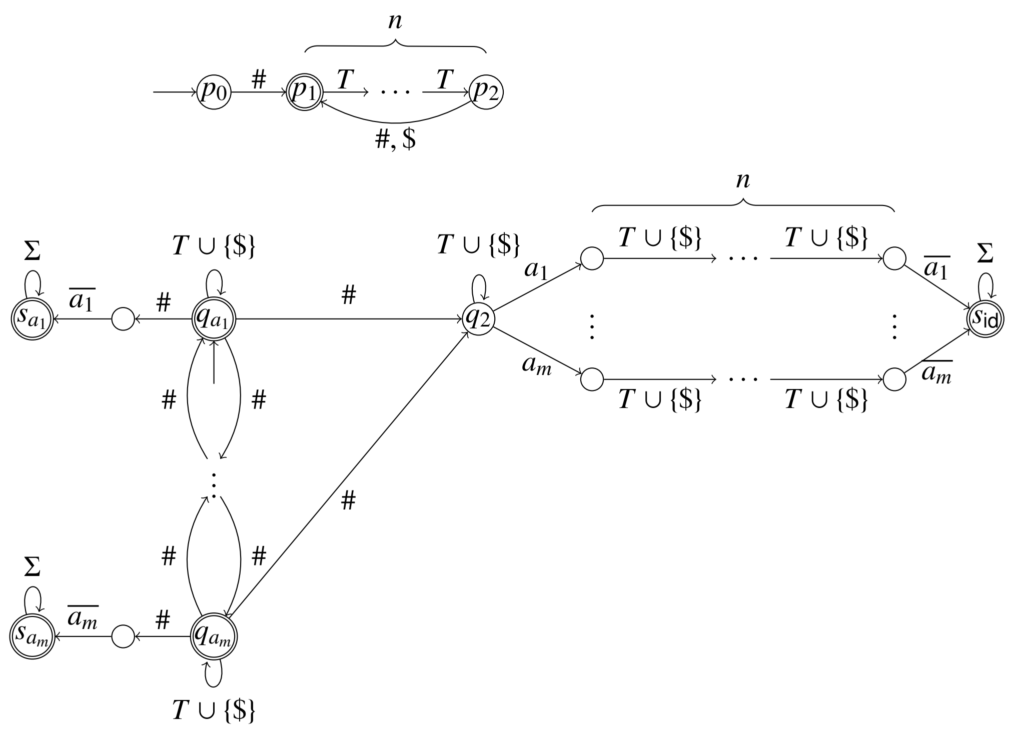


Figure 4.3: NBA $\mathcal{A}$ (above) and $\mathcal{B}$ (below) to encode the interaction in the corridor tiling game.

Proof. For the first part, the strategy for Duplicator is simply to wait until Spoiler reads $w' \in T^n$. If $w'(1) \neq t$ then Duplicator loops in q_t by reading w and then goes to s_t by reading $\#w'(1)$. From the accepting sink, it is not hard to see that Duplicator can play accordingly and win the play. In the case where $w'(1) = t$, Duplicator pops w by looping in q_t. The play then proceeds to the configuration (4.6).

For the second part, the strategy is as follows. Initially, Duplicator pops # from the buffer and goes to $q_{t'}$, i.e. she proceeds to

$$(p_1, w', q_{t'}, \top, \mathsf{S}). \tag{4.8}$$

If Spoiler responds to this by reading # then we have the desired result. Otherwise, Spoiler reads \$ and in this case, Duplicator skips her turn for the next n rounds. Hence we reach some configuration $(p_1, w'\$w'', q_{t'}, \bot, \mathsf{D})$ where $w'' \in T^n$.

If there exists $i \in \{1, \ldots, n-1\}$ such that $w'(i) \neq w''(i)$ then Duplicator goes to s_{id}: she pops $w'(1), \ldots, w'(i-1)$ by going to q_2, and then pops $w'(i), \ldots, w'(n), \$, w''(1), \ldots, w''(i)$ by going to s_{id}. From s_{id}, Duplicator can play accordingly and win the play.

Otherwise, we have $w' = w''$. In this case, Duplicator pops $w'\$$ by looping in $q_{t'}$. Hence we reach the configuration (4.8) again.

Duplicator repeats this procedure indefinitely. If Spoiler never reads #, the play reaches the configuration as in (4.8) infinitely often, and in this case Duplicator wins. Otherwise, Spoiler eventually reads # and we reach the configuration (4.7). □

The first part of this lemma shows that after Spoiler reads $w\#$, where $w \in T^n$, by being in some state q_t where $t \in T$, Duplicator forces Spoiler to produce a word $w' \in T^n$, the tiling of the next row, in which the first tile is t. The second part shows that Duplicator can choose any tile $t' \in T$ as the first tile of the next row, i.e. by moving to $q_{t'}$, and then force Spoiler to eventually stop repeating the current tiling by looping in $q_{t'}$ to mimic the repeated tiling. Spoiler eventually starts producing the tiling of the next row since otherwise he will lose because Duplicator forms an accepting run.

Now to illustrate what Spoiler can do in the game $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, consider the play from some configuration $(p_1, w\#, q_t, \top, \mathsf{D})$ where $w \in T^n$ or from any configuration of the form $(p_1, \hat{w}\$w\$w \ldots \#, q_t, \top, \mathsf{D})$ where $\hat{w}$ is a suffix of w. If Spoiler continues by reading a tiling w' in which $w'(1) = t$ and repeating it, i.e. reading $w'\$w'\$\ldots$, then Duplicator will never reach any accepting sink in $\mathcal{B}$. She is indeed forced to eventually pop # from the buffer and move to some state $q_{t'}$. We formulate this observation as follows.

Lemma 4.3.21. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ as in Figure 4.4 and a play in $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$ from some configuration*

$$(p_1, \widetilde{w}\#, q_t, b, \mathsf{D})$$

where $w \in T^n$, $\widetilde{w} \in \mathsf{suffix}(w) \cdot (\$\, w)^*$, $t \in T$, and $b \in \{\top, \bot\}$. For any $w' \in T^n$ where $w'(1) = t$, Spoiler *has a strategy such that either* Duplicator *loses or the play proceeds to some configuration*

$$(p_1, \widetilde{w}'\#, q_{t'}, b', \mathsf{D}) \tag{4.9}$$

$\widetilde{w}' \in \mathsf{suffix}(w') \cdot (\$w')^*$, $t' \in T$, and $b' \in \{\top, \bot\}$.

Proof. The strategy for Spoiler is as follows. He reads $w'\$w'\$w' \ldots$ indefinitely until Duplicator pops # from the buffer. If Duplicator never does so then Spoiler wins because he forms an accepting run. Hence let us assume that Duplicator eventually pops # from the buffer by going to some state q. We have $q \neq s_t$ because $w'(1) = t$. We also have $q \neq s_{t'}$ for all $t' \in T$, $t' \neq t$, since there is only one # in the buffer and we need at least two to reach such $s_{t'}$ from q_t. Moreover, $q \neq s_{\mathsf{id}}$ because for all $i > 0$, the i-th and the $i + (n + 1)$-th letters of $w'\$w'\$w' \ldots$ are identical. Hence either q is one of the main states, i.e. $q = q_{t'}$ for some $t' \in T$, or some other non-sink state. In the second case, Spoiler wins by keep reading $w'\$w'\$w' \ldots$ since we have seen that Duplicator will never reach an accepting sink and form an accepting run. In the first case, Spoiler does as follows. If Duplicator pops # when Spoiler reads \$ or the k-th letter of w', Spoiler continues reading until the last letter of w' and then reads #. Hence we reach a configuration as in (4.9). □

This lemma basically shows that by repeating the tiling of the current row, Spoiler forces Duplicator to choose the first tile of the next row. Duplicator cannot reach any accepting sink. Note that the main states $q_{a_1}, \ldots, q_{a_m}$ in $\mathcal{B}$ intuitively correspond to the tiles that can be chosen by Duplicator. The move of Duplicator from q_t to $q_{t'}$ corresponds to the move of Starter that chooses the tile t' for the next row, after choosing t for the current row.

We can slightly extend the automata $\mathcal{A}$, $\mathcal{B}$ to fully encode the corridor tiling game. To encode the vertical and horizontal mismatches, we simply extend Duplicator's automaton with the components $\mathcal{B}_{\mathsf{hor}}$ and $\mathcal{B}_{\mathsf{ver}}$ that we have illustrated in Figure 4.2. We extend the automaton $\mathcal{B}$ such that Spoiler is forced to produce a tiling that matches horizontal and vertically since otherwise Duplicator can reach the accepting sink s_{hor} or s_{ver}. Furthermore, we can also force Duplicator to choose the first tile that matches vertically. We simply restrict the #-transition from the main states $q_{a_1}, \ldots, q_{a_m}$. For any main state q_t, $t \in T$, we have

$$q_t \xrightarrow{\#} q_{t'} \text{ iff } (t, t') \in V.$$

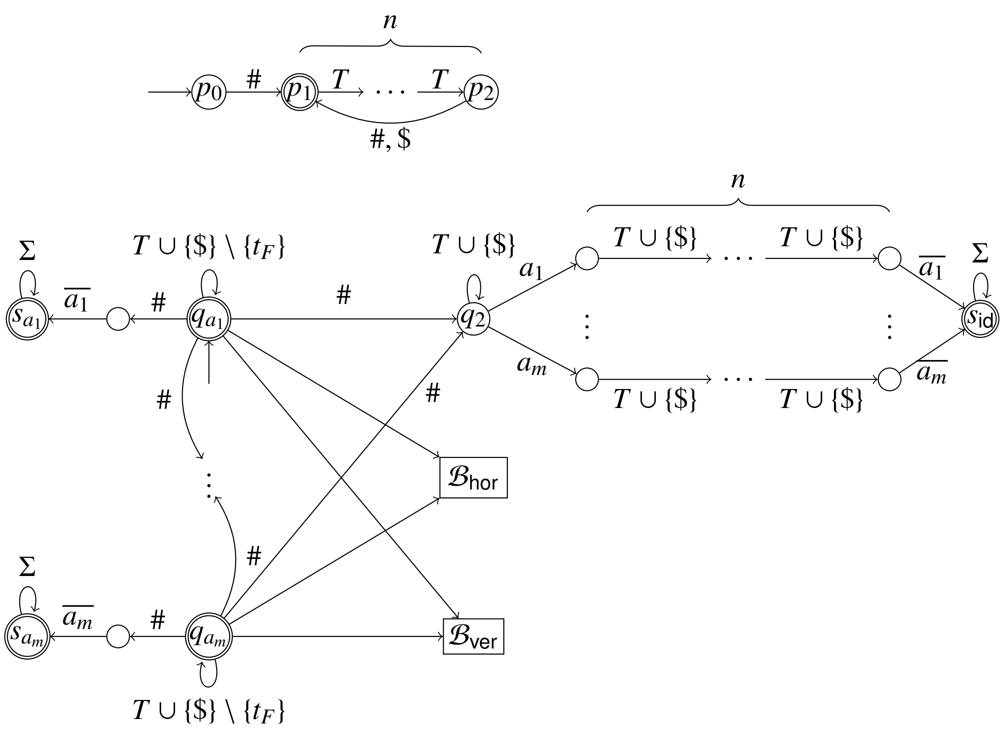


Figure 4.4: NBA $\mathcal{A}$ (above) and $\mathcal{B}$ (below) for the corridor tiling game.

Hence Duplicator can choose t' as the first tile of the next row if t' is vertically matched with the first tile t of the current row. Moreover, we can also encode the situation where Completer immediately wins when the final tile is used. We simply disallow Duplicator to pop the final tile from the main states $q_{a_1}, \ldots, q_{a_m}$. Hence once Spoiler produces a row with a final tile, he can repeat it forever and win the play. Duplicator cannot mimic such a move because she cannot pop t_F from the buffer. We illustrate such extended automata in Figure 4.4.

By considering such automata $\mathcal{A}$, $\mathcal{B}$, we can extend the property described in Lemma 4.3.20. Intuitively, after Spoiler produces a tiling $w \in T^n$, if w does not contain the final tile, Duplicator can force Spoiler to produce a word $w' \in T$, the tiling of next row, that is horizotally compatible and also vertically compatible with w. We lift the property in Lemma 4.3.20 as follows.

Lemma 4.3.22. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ as in Figure 4.4 and a play in $\mathcal{G}^{\omega}(\mathcal{A},\mathcal{B})$ from configuration*

$$(p_1, w\#, q_t, b, \mathsf{D})$$

where $w \in T^n \cup \{\epsilon\}$, $t \in T$, and $b \in \{\top, \bot\}$.

- *If $w(i) \neq t_F$ for all $i \in \{1, \ldots, n\}$,* Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to some configuration*

$$(p, \#w', q_t, \bot, \mathsf{D}) \tag{4.10}$$

  where $w' \in T^n$ and the following holds.

  - *$w'(1) = t$,*
  - *$(w'(i), w'(i+1)) \in H$ for all $i \in \{1, \ldots, n-1\}$, and*

  - $(w(i), w'(i)) \in V$ *for all* $i \in \{1, \ldots, n\}$.

- *From (4.10), if* $w'(i) \neq t_F$ *for all* $i \in \{1, \ldots, n\}$ *then for any* $t' \in T$ *where* $(t, t') \in T$, Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to*

$$(p_1, w'\#, q_{t'}, \top, \mathsf{D}).$$

Proof. For the first part, the strategy for Duplicator is the same as in the first part of Lemma 4.3.20. He simply waits until Spoiler reads $w' \in T^n$. Additionally, we have $(w'(i), w'(i+1)) \in H$ for all $i \in \{1, \ldots, n-1\}$ and $(w(i), w'(i)) \in V$ for all $i \in \{1, \ldots, n\}$ since otherwise Duplicator can go to the accepting sink s_{hor} or s_{ver}. Moreover, Duplicator can pop w by looping in q_t since w does not contain t_F.

For the second part, the strategy for Duplicator is the same as in the second part of Lemma 4.3.20. □

We can also extend Lemma 4.3.21 in a similar way. Intuitively, after Duplicator announces the first tile and forces Spoiler to produce a new row, if Spoiler produces a tiling $w' \in T^n$ that is vertically and horizontally compatible with the current row then whenever w' contains the final tile, Spoiler wins by repeating w' for the rest of the play. In the case where w' does not contain the final tile, by repeating w', Spoiler forces Duplicator to eventually announce the first tile of the next row. We lift the property in Lemma 4.3.21 as follows.

Lemma 4.3.23. *Consider the two NBA* $\mathcal{A}$, $\mathcal{B}$ *as in Figure 4.4 and a play in* $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ *from configuration*

$$(p_1, \widetilde{w}\#, q_t, b, \mathsf{D})$$

where $w \in T^n$, $\widetilde{w} \in \mathsf{suffix}(w) \cdot (\$\, w)^*$, $t \in T$, *and* $b \in \{\top, \bot\}$. *For any* $w' \in T^n$ *with*

- $w'(1) = t$,
- $(w'(i), w'(i+1)) \in H$ *for all* $i \in \{1, \ldots, n-1\}$, *and*
- $(w(i), w'(i)) \in V$ *for all* $i \in \{1, \ldots, n\}$,

the following holds.

- *If there exists* $i \in \{1, \ldots, n\}$ *such that* $w'(i) = t_F$ *then* Spoiler *has a strategy to win the play.*
- *If for all* $i \in \{1, \ldots, n\}$, $w(i) \neq t_F$, *then* Spoiler *has a strategy such that either* Duplicator *loses or the play proceeds to some configuration*

$$(p_1, \widetilde{w}'\#, q_{t'}, b', \mathsf{D})$$

  where $\widetilde{w}' \in \mathsf{suffix}(w') \cdot (\$w')^*$, $(t, t') \in V$, *and* $b' \in \{\top, \bot\}$.

Proof. For the first part, let $i_0 \in \{1, \ldots, n\}$ such that $w'(i_0) = t_F$. The strategy for Spoiler is simpler than the one in Lemma 4.3.21. He reads $w'\$w'\$\ldots$ for the rest of the play. If Duplicator never pops # then Spoiler wins. If Duplicator eventually pops # then, as we have seen in the proof of Lemma 4.3.21, she does not go to the accepting s_{id} or any $s_{t'}$, $t \in T$, and loses if she moves to some other states beside the main ones. Additionally, in this case, she also does not pop # by going to the accepting sink s_{hor} or s_{ver} because

$(w'(i), w'(i+1)) \in H$ for all $i \in \{1, \ldots, n-1\}$ and $(w(i), w'(i)) \in V$ for all $i \in \{1, \ldots, n\}$. Hence let us assume that Duplicator goes to some main state $q_{t'}$. By definition of $\mathcal{B}$, we have $(t, t') \in V$. Moreover, since $w'(i_0) = t_F$ and there is no t_F-successor from $q_{t'}$, Duplicator cannot pop $w'(i_0)$ from the buffer. She will not form an accepting run from $q_{t'}$. Since Spoiler forms an accepting run and Duplicator does not, Spoiler wins the play.

For the second part, the strategy for Spoiler is the same as the one in Lemma 4.3.21. □

We can use Lemma 4.3.22 and Lemma 4.3.23 to show that Completer wins $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, a_1, t_F, n)$ iff Spoiler wins $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$. We translate the winning strategy of Completer to Spoiler and Starter to Duplicator.

Theorem 4.3.24. *Given a tiling system $\mathcal{T} = (T, H, V)$, initial and final tiles $t_{1,1}, t_F \in T$, and a width $n > 0$, we can construct in polynomial time two NBA $\mathcal{A}, \mathcal{B}$ such that* Completer *wins $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_{1,1}, t_F, n)$ iff* Spoiler *wins $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$.*

Proof. Consider $\mathcal{A}, \mathcal{B}$ as given in Figure 4.4. Suppose Completer wins $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_{1,1}, t_F, n)$. If in the initial round, he proceeds from the initial configuration $(t_{1,1}, \mathsf{Comp})$ to

$$(t_{1,1}, \ldots, t_{n,1}, \mathsf{St}) \tag{4.11}$$

then in $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, Spoiler proceeds as follows. Initially, he reads # and proceeds to $(p_1, \#, q_{t_{1,1}}, \top, \mathsf{D})$. If one of $t_{1,1}, \ldots, t_{n,1}$ is the final tile then Spoiler follows the strategy as described in the first part of Lemma 4.3.23 to push $w_1 = t_{1,1}, \ldots, t_{n1}$ and win the play. Otherwise, he follows the strategy as in the second part to push $w_1 = t_{1,1}, \ldots, t_{n,1}$ and according to the lemma, he either wins or the play proceeds to

$$(p_1, \widetilde{w_1}\#, q_{t_{1,2}}, b_1, \mathsf{D}) \tag{4.12}$$

where $\widetilde{w_1} \in \mathsf{suffix}(w_1)\cdot(\$w_1)^*$, $(t_{1,1}, t_{1,2}) \in V$, and $b_1 \in \{\top, \bot\}$. Spoiler then considers what Completer would do in the tiling game if from (4.11), Starter continues to $(t_{1,2}, \mathsf{Comp})$. If Completer responds to this by going to $(t_{1,2} \ldots t_{n,2}, \mathsf{St})$ then from (4.12), Spoiler again follows the strategy as described in Lemma 4.3.23 to push $w_2 = t_{1,2}, \ldots, t_{n,2}$. We repeat this procedure indefinitely. Since Completer wins the tiling game, either the final tile is eventually used or Starter gets stuck because she cannot choose a first tile that matches vertically. In the first case, Spoiler wins by considering the strategy from the first part of Lemma 4.3.23 and in the second case, Spoiler wins by considering the strategy from the second part of Lemma 4.3.23. Spoiler wins $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$.

Now suppose Starter wins $\mathcal{G}_{\mathsf{tile}}(\mathcal{T}, t_{1,1}, t_F, n)$. In the first round of $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$, Spoiler initially will read # and the play will proceed to $(p_1, \#, q_{t_{1,1}}, \bot, \mathsf{D})$. Duplicator then follows the strategy as described in the first part of Lemma 4.3.22. According to Lemma 4.3.22, either she wins or the play eventually proceeds to some configuration

$$(p_1, \#w_1, q_{t_{1,1}}, \bot, \mathsf{D}) \tag{4.13}$$

where $w_1 = t_{1,1} \ldots t_{n,1}$ and $(t_{i,1}, t_{i+1,1}) \in H$ for all $i \in \{1, \ldots, n-1\}$. Note that for all $i \in \{1, \ldots, n\}$, we have $t_{i,1} \neq t_F$, since otherwise Completer would have won the tiling game. Duplicator then considers what Starter would do in the tiling game if from the initial configuration, Completer proceeds to $(t_{1,1} \ldots t_{n,1}, \mathsf{St})$. If Starter responds to this by going to $(t_{1,2}, \mathsf{Comp})$ then from (4.13), Duplicator follows the strategy as described in the second part of Lemma 4.3.22 by moving to $q_{t_{1,2}}$. According to the lemma, either Duplicator wins or the play eventually proceeds to

$$(p_1, w_1\#, q_{t_{1,2}}, \top, \mathsf{D}). \tag{4.14}$$

Duplicator then follows the strategy as described in Lemma 4.3.22 again. She repeats this procedure for the rest of the play. Hence either eventually Duplicator wins by following the strategy as described in Lemma 4.3.22 or the play reaches some configuration as in (4.14) infinitely often. In such a case, Duplicator forms an accepting run. She also wins the play. Duplicator wins $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$. □

Note that the size of the automata $\mathcal{A}$ and $\mathcal{B}$ in Figure 4.4 is polynomially larger than the tiling system $\mathcal{T}$ and the corridor width n. We also have a polynomial-time reduction from the corridor tiling game to the buffered simulation game with one unbounded buffer. Since solving the corridor tiling game is EXPTIME-hard, deciding whether Duplicator wins $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ is also EXPTIME-hard. Hence we have the following corollary.

Corollary 4.3.25. *Deciding $\sqsubseteq^\omega$ is EXPTIME-hard.*

4.3.3 Upper Bound for Deciding $\sqsubseteq^\omega$

Corollary 4.3.25 shows that deciding buffered simulation with one unbounded buffer is EXPTIME-hard. However, it is still not clear whether the problem is decidable. One might get tempted to consider a technique similar to what we have used for deciding $\sqsubseteq^\omega_{\mathsf{Flush}}$. This, unfortunately, is not possible. It is true that in the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ and $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, Duplicator can skip her turn as long as she wants. However unlike $\mathcal{G}^\omega_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$, in the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$, Duplicator can store some letters in the buffer as much as she wants after she moves the pebble. Recall that the PSPACE-algorithm for deciding $\sqsubseteq^\omega_{\mathsf{Flush}}$ heavily relies on the fact that once Duplicator moves, the play always proceeds to some configuration with an empty buffer. Since there are only finitely many such configurations, we can check for each of them, whether it is winning for Spoiler. This technique however is not suitable for the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$. In the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$, after Duplicator moves, there might be some letters left in the buffer. There are infinitely many possible configurations that can be reached after Duplicator's move.

Nevertheless we will show that the problem of deciding $\sqsubseteq^\omega$ is indeed still decidable. We will use a more involved technique. We will even show that the lower bound in Corollary 4.3.25 is tight.

Before we present the algorithm for deciding $\sqsubseteq^\omega$, let us recall the equivalence relation $\approx$ defined in Definition 2.3.6. We will use this equivalence relation to show an important characterisation of accepting runs in $\mathcal{A}$, $\mathcal{B}$ which will be the key of our algorithm. The characterisation is similar to the characterisation of infinite words that are used to show complementation of Büchi automata [Büc62].

First recall that in Definition 2.3.6, the equivalence relation $\approx$ is defined with respect to a single automaton $\mathcal{A}$. Two words $u, v \in \Sigma^*$ are equivalent, i.e. $u \approx v$, if they both behave the same in the automaton $\mathcal{A}$. We will consider a natural extension of $\approx$ that is defined with respect to a pair of automata $\mathcal{A}$, $\mathcal{B}$ instead of a single automaton $\mathcal{A}$. Intuitively, two finite words u, v are equivalent if they both behave the same in the automata $\mathcal{A}$, $\mathcal{B}$: if by reading u we can go from state p to p' in $\mathcal{A}$ and q to q' in $\mathcal{B}$ then we can also do the same by reading v and if by reading u from p to p' or from q to q' we see a final state, we also see a final state by reading v. We refrain from giving a new definition for the relation $\approx$.

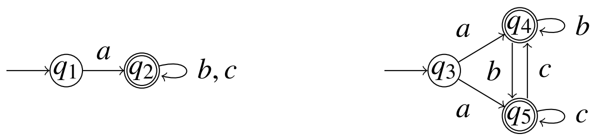
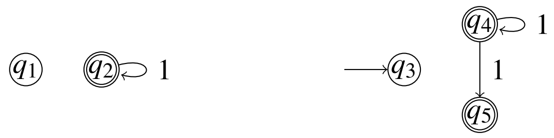


Figure 4.5: NBA $\mathcal{A}$, $\mathcal{B}$ for Example 4.3.26.

For the rest of this subsection, we will simply assume that the automata $\mathcal{A}$ and $\mathcal{B}$ share the same state space and only differ in their initial states.

Example 4.3.26. Consider the two automata $\mathcal{A}$, $\mathcal{B}$ in Figure 4.5 where $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$, $\mathcal{B} = (Q, \Sigma, \delta, q_3, F)$, $Q = \{q_1, \dots, q_5\}$, $\Sigma = \{a, b, c\}$, and $F = \{q_2, q_4, q_5\}$. We have $bb \approx bc$ since by reading bb or bc, we either make an accepting cycle from q_2 or from q_4, or a path from q_4 to q_5 that goes through an accepting state. The equivalence class of bb in fact consists of any word over b or c that starts with b, i.e. $[bb] = b \cdot \{b, c\}^*$. Hence the transition profile of the equivalence class $[bb]$ is as follows.



In the preliminary chapter, we have shown that the equivalence classes induced by the relation $\approx$ can be used to finitely characterise Σ^ω. By Proposition 2.3.12, for every infinite word $w \in \Sigma^\omega$, there exists a pair of proper equivalence classes $[u], [v] \in \Sigma^*/\approx$ such that $w \in [u][v]^\omega$, and recall that by proper it means the pair $[u]$, $[v]$ satisfies $[uv] = [u]$ and $[vv] = [v]$.

Now suppose we have an infinite run ρ in $\mathcal{A}$ or $\mathcal{B}$ from $p \in Q$. Since the numbers of states in $\mathcal{A}$ and $\mathcal{B}$ are finite, there must be a state $p' \in Q$ that appears infinitely often in ρ. Hence for every accepting run ρ, we can see it as a run that initially goes from p to p' and then is followed by infinitely many cycles over p'. Moreover if ρ is accepting, we can see ρ as a run that initially goes from a starting state p to some state p' followed by infinitely many accepting cycles over p', i.e.

$$\rho = p \xrightarrow{w_0} p' \xrightarrow[F]{w_1} p' \xrightarrow[F]{w_2} p' \dots. \tag{4.15}$$

Now one interesting property holds if ρ is an accepting run over some word $w \in [u][v]^\omega$ where $[u], [v]$ is a proper pair. In such a case, we can factorise ρ into the form (4.15) where $w_0 \in [u]$ and $w_1, w_2, \dots \in [v]$. This is simply because the pair $[u]$, $[v]$ is proper.

Lemma 4.3.27. *Let ρ be an accepting run in $\mathcal{A}$, $\mathcal{B}$ from $p \in Q$ over a word $w \in \Sigma^\omega$. If $w \in [u][v]^\omega$ and $[u], [v]$ is a proper pair then we can factorise ρ into*

$$\rho = p \xrightarrow{w_0} p' \xrightarrow[F]{w_1} p' \xrightarrow[F]{w_2} p' \dots \tag{4.16}$$

where $w_0 \in [u]$ and $w_1, w_2, \dots \in [v]$.

Proof. Suppose we have an accepting run ρ in $\mathcal{A}$ or $\mathcal{B}$ from $p \in Q$ over $w \in [u][v]^\omega$ where the pair $[u], [v]$ is proper. Since $w \in [u][v]^\omega$, the word w can be factorised into $w = w_0 w_1 \dots$ where $w_0 \in [u]$ and $w_1, w_2 \dots \in [v]$. This implies that we can also factorise the run ρ into

$$\rho = p \xrightarrow{w_0} p_1 \xrightarrow{w_1} p_2 \xrightarrow{w_2} p_3 \dots \tag{4.17}$$

for some $p_1, p_2, \ldots \in Q$. Since ρ is accepting, there exist infinitely many i such that $p_i \xrightarrow[F]{w_i} p_{i+1}$. Moreover since the numbers of the states in $\mathcal{A}$ and $\mathcal{B}$ are finite, there exists a state $p' \in Q$ and infinitely many j such that $p_j = p'$. Hence we can find infinitely many indices $i_1 < i_2 < \ldots$ such that

$$\rho = p \xrightarrow{w_0 \ldots w_{i_1}} p' \xrightarrow[F]{w_{i_1+1} \ldots w_{i_2}} p' \xrightarrow[F]{w_{i_2+1} \ldots w_{i_3}} p' \ldots.$$

Note that the pair $[u], [v]$ is proper, that is, $[uv] = [u]$ and $[vv] = [v]$. Since $w_1, w_2, \ldots \in [v]$ and $[vv] = [v]$, we have $w_i \ldots w_j \in [v]$ for all $0 < i < j$. Moreover, since $w_0 \in [u]$ and $[uv] = [u]$, we additionally have $w_0 \ldots w_i \in [u]$ for all $i > 0$. Hence $w_0 \ldots w_{i_1} \in [u]$ and $w_{i_k+1} \ldots w_{i_{k+1}} \in [v]$ for all $k > 0$. □

We can even simplify this property with Proposition 2.3.12. Recall that by Proposition 2.3.12, for any infinite word $w \in \Sigma^\omega$, there exists a proper pair $[u]$, $[v] \in \Sigma^*/\approx$ such that $w \in [u][v]^\omega$ [Büc62, FV09]. Hence the premise in Lemma 4.3.27 is always true and we have the following corollary.

Corollary 4.3.28. *For any accepting run ρ in $\mathcal{A}$, $\mathcal{B}$ from $p \in Q$, the run ρ can be factorised into*

$$\rho = p \xrightarrow{w_0} p' \xrightarrow[F]{w_1} p' \xrightarrow[F]{w_2} p' \ldots \tag{4.18}$$

where $w_1, w_2, \ldots \in [w_1]$ and the pair $[w_0], [w_1]$ is proper.

Proof. Let ρ be an accepting run from $p \in Q$ and $w = \mathsf{word}(\rho)$. By Proposition 2.3.12, there exists a proper pair $[u], [v]$ such that $w \in [u][v]^\omega$. By Lemma 4.3.27, we can factorise ρ into $\rho = p \xrightarrow{w_0} p_1 \xrightarrow[F]{w_1} p_2 \xrightarrow[F]{w_2} p_3 \ldots$ where $w_0 \in [u]$ and $w_1, w_2, \ldots \in [v]$. Hence we have the desired result. □

Now let us formally call a finite run that ends in a cycle a *lasso*.

Definition 4.3.29. A finite run $r = p_1 a_1 p_2 \ldots a_{n-1} p_n$ with $n > 1$, is a lasso if there exists $i \in \{1, \ldots, n-1\}$ such that $p_i = p_n$.

Intutively, a finite run r is a lasso if r can be factorised into

$$r = p \xrightarrow{u} p' \xrightarrow{v} p'$$

where $u \in \Sigma^*$ and $v \in \Sigma^+$. In such a case, we will simply call r a lasso over the pair of words u, v. A lasso basically consists of two parts: the initial part and the cycle part. If the cycle part of the lasso goes through an accepting state, let us call the lasso to be *accepting*, and if the words in the initial and cycle parts belong to some proper pair of equivalence classes, let us call the lasso to be *proper*.

Example 4.3.30. Consider again the two automata $\mathcal{A}$, $\mathcal{B}$ in Figure 4.5. The finite runs $r_1 = q_1 a q_2 b q_2 c q_2$ and $r_2 = q_3 a q_4 b q_5 c q_4$ are accepting lassos over a, bc since they can be factorised into

$$r_1 = q_1 \xrightarrow{a} q_2 \xrightarrow[F]{bc} q_2,$$

$$r_2 = q_3 \xrightarrow{a} q_4 \xrightarrow[F]{bc} q_4.$$

Moreover note that the pair $[a], [bc]$ is proper. We have $a \approx abc$ since by reading a or abc, we either go from q_1 to q_2, q_3 to q_4, or q_3 to q_5, through an accepting state. We also have $bcbc \approx bc$ since in Example 4.3.26, we have seen that every finite words over b, c that are started with b are equivalent to each other. Hence r_1 and r_2 are indeed proper accepting lassos over a, bc.

Corollary 4.3.28 shows that any accepting run always starts with a proper accepting lasso. Intuitively, an accepting run can be seen as a proper accepting lasso that repeats the cycle part infinitely often. By repeating the cycle part, we mean forming an accepting cycle over a word that is not necessarily the same, but equivalent with respect to $\approx$. For example, the accepting run ρ in (4.18) is a proper accepting lasso over w_0, w_1 that repeats the cycle part infinitely often by reading $w_2, w_3, \ldots \approx w_1$.

For any NBA $\mathcal{A}$, a proper accepting lasso characterises the accepting words from some state. If there exists a proper accepting lasso from p, suppose over u, v, then for any word $w \in [u][v]^\omega$, the word w is accepted from p. The converse also holds: if there is no proper accepting lasso over u, v from p then for any word $w \in [u][v]^\omega$, the word w is not accepted from p. This idea is similar to the characterisation of infinite words with respect to some NBA $\mathcal{A}$ by using equivalence classes $\Sigma^*/\approx$ [Büc62, FV09].

Proposition 4.3.31. *Let $u, v \in \Sigma^*$ be two finite words where the pair $[u], [v]$ is proper.*

- *If there exists an accepting lasso r where*

$$r = p \xrightarrow{u} p' \xrightarrow[F]{v} p' \tag{4.19}$$

then for any $w \in [u][v]^\omega$, there is an accepting run over w from p.

- *If there is no accepting lasso as in (4.19) then for any $w \in [u][v]^\omega$, there is no accepting run over w from p.*

Proof. For the first part, let $w \in [u][v]^\omega$. Hence w can be factorised into $w = w_0 w_1 \ldots$ where $w_0 \in [u]$ and $w_1, w_2, \ldots \in [v]$. Suppose we have an accepting lasso as in (4.19). Since $w_0 \approx u$ and $w_1, w_2, \ldots \approx v$, we can consider the accepting run ρ where

$$\rho = p \xrightarrow{w_0} p' \xrightarrow[F]{w_1} p' \xrightarrow[F]{w_2} p' \ldots.$$

over w.

For the second part, let us show it by contradiction. Suppose there is no accepting lasso as in (4.19), but there is an accepting run ρ over w where $w \in [u][v]^\omega$. By Lemma 4.3.27, the run can be factorised into

$$p \xrightarrow{w_0} p' \xrightarrow[F]{w_1} p' \xrightarrow[F]{w_2} p' \ldots$$

where $w_0 \in [u]$ and $w_1, w_2 \ldots \in [v]$. Since $w_0 \approx u$ and $w_1 \approx v$, we also have an accepting lasso over u, v from p to p' which contradicts our initial assumption. □

We will consider this observation to reduce the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ into a simpler game.

Reduction to a Lasso Game

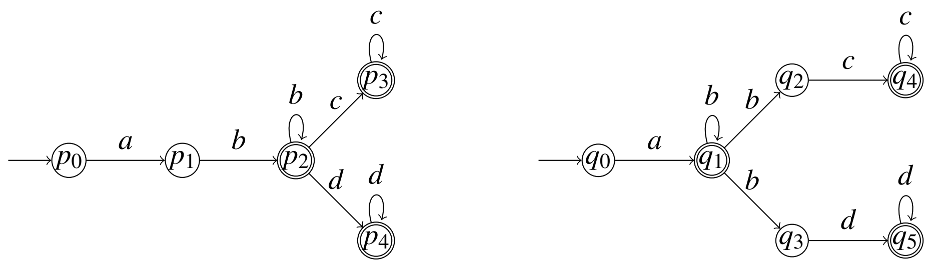
First let us look at some possible winning strategy for Duplicator in $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$. In the buffered simulation game $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$, Duplicator is equipped with an unbounded buffer. She can store as many letters as she wants to foresee Spoiler's run. Hence one possible strategy for Duplicator is to wait indefinitely until she is sure that Spoiler indeed can form an accepting run. She waits until Spoiler forms a proper accepting lasso. If Spoiler never forms a proper accepting lasso, by Lemma 4.3.27, he also does not form an accepting run. Duplicator wins the play by waiting forever. However if Spoiler eventually forms a proper accepting lasso, suppose over w_0, w_1 then Duplicator needs to check whether she can also form a proper accepting lasso over w_0, w_1 from the initial state of $\mathcal{B}$. If Duplicator cannot form one then she will lose the play. Spoiler can keep repeating the accepting loop forever and form some accepting run over $w \in [w_0][w_1]^\omega$. By the second part of Proposition 4.3.31, there is no accepting run over w that can be formed by Duplicator from the initial state of $\mathcal{B}$, and hence she loses the play. However if there is such an accepting lasso over w_0, w_1 then Duplicator should wait a bit more and not immediately form a corresponding accepting lasso. Duplicator should wait until Spoiler repeats the cycle part once, i.e. until Spoiler forms

$$p_0 \xrightarrow{w_0} p_1 \xrightarrow[F]{w_1} p_1 \xrightarrow[F]{w_2} p_1 \tag{4.20}$$

where $w_2 \approx w_1$. After Spoiler forms such an extended lasso, Duplicator pops w_0w_1 by forming a corresponding one over w_0, w_1, lets w_2 stay in the buffer, and waits again until Spoiler forms another proper accepting lasso from p_1 and repeats the procedure.

Now one might wonder whether waiting until Spoiler repeats the cycle part is necessary. It is indeed necessary to wait until Spoiler repeats the cycle part and we will show this in the following example.

Example 4.3.32. Let $\mathcal{A}$, $\mathcal{B}$ be the following two NBA.



According to the strategy that we have described before, to win the game $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$, Duplicator should wait until Spoiler forms a proper accepting lasso. In this case, initially Duplicator waits until Spoiler either reads *abb*, *abcc*, or *abdd*. Suppose Spoiler reads *abb*. If Duplicator immediately flushes the buffer then the play proceeds to the configuration $(p_2, \epsilon, q_1, \top, \mathsf{S})$. From this configuration, Spoiler can continue by reading c or d and make Duplicator get stuck in q_1. Hence Duplicator loses. This, however, will not happen if Duplicator additionally waits until Spoiler repeats the cycle part, i.e. until Spoiler reads *abbb*. In this case, Duplicator forms a corresponding lasso by reading *abb* and proceeding to the configuration $(p_2, b, q_1, \top, \mathsf{S})$. From this configuration, Duplicator then waits again until Spoiler forms a proper accepting lasso and repeats the cycle part, i.e. until Spoiler either reads *bb*, *ccc*, or *ddd*. Duplicator again can respond to this by forming a corresponding lasso, that is, by reading *bb*, *bcc*, or *bdd* respectively and proceeding to

	Round 1	Round 2	Round 3
S chooses:	$p_0 \xrightarrow{u_1} p_1 \xrightarrow[F]{v_1} p_1 \xrightarrow[F]{v_1'} p_1$	$p_1 \xrightarrow{u_2} p_2 \xrightarrow{v_2} p_3 \xrightarrow[F]{v_2'} p_3$	$\cdots$
D chooses:	$q_0 \xrightarrow{u_1} q_1 \xrightarrow[F]{v_1} q_1$	$q_1 \xrightarrow{v_1' u_2} q_2 \xrightarrow[F]{v_2} q_2$	$\cdots$

Figure 4.6: A game where each of the players alternatingly chooses a lasso.

the configuration $(p_2, b, q_1, \top, \mathsf{S})$, $(p_3, c, q_4, \top, \mathsf{S})$, or $(p_4, d, q_5, \top, \mathsf{S})$. By repeating such a procedure, Duplicator will never get stuck and win the play.

Note that if Duplicator plays according to the strategy as described before, there are three possibilities: first, at one point she waits for the rest of the play because Spoiler does not form a proper accepting lasso and repeat the cycle part as in (4.20), second, Spoiler infinitely often forms a proper accepting lasso and repeats the cycle part, and Duplicator always forms a corresponding lasso, or third, at one point Spoiler forms a proper accepting lasso and repeats the cycle part, but Duplicator cannot form a corresponding one. In the first case, Duplicator wins because by Lemma 4.3.27, we know that Spoiler does not form an accepting run. In the second one, Duplicator also wins because she forms an accepting lasso infinitely often and hence forms an accepting run. In the third case, Duplicator loses the play because Spoiler can extend the lasso into an accepting run that cannot be mimicked by her. This intuitively allows us to reduce the game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ into a game where both of the players move alternatingly by choosing a proper accepting lasso. In every round $i > 0$, Spoiler forms a proper accepting lasso over u_i, v_i and repeats the cycle part once by reading $v_i' \approx v_i$. Duplicator responds to this by forming a corresponding accepting lasso over u_i, v_i after reading v_{i-1}'. If one of the players cannot do so then the opponent wins. We illustrate how such a game proceeds in Figure 4.6.

In such a game, the number of configurations is still infinite. There are infinitely many possible lassos that can be chosen by Spoiler. However, we can finitely represent lassos by using the equivalence classes from $\Sigma^*/{\approx}$. First note that in the transition profile of some equivalence class $[u]$, if there is an edge (p,p') then there is a path from p to p' by reading u. Moreover, if the edge is labeled with 1 then there is such a path that goes through an accepting state. Let us denote with $p \xrightarrow{[u]} q$ if there exists an edge from p to q in the transition profile of $[u]$ and $p \xrightarrow[F]{[u]} q$ if the edge is labeled with 1. Hence instead of choosing a proper accepting lasso from p, Spoiler can choose a proper pair of equivalence classes $[u], [v] \in \Sigma^*/{\approx}$ and a state p' such that $p \xrightarrow{[u]} p'$ and $p' \xrightarrow[F]{[v]} p'$.

We can equivalently consider a game where in each round the players do not choose a lasso, but its representation. In every round $i > 0$, Spoiler chooses a pair of proper equivalence classes $[u_i], [v_i]$ and a state p_i such that

$$p_{i-1} \xrightarrow{[u_i]} p_i \xrightarrow[F]{[v_i]} p_i$$

and Duplicator chooses a state q_i such that

$$q_{i-1} \xrightarrow{[v_{i-1}][u_i]} q_i \xrightarrow[F]{[v_i]} q_i.$$

Let us call such a game *lasso game*. It is formally defined as follows.

Definition 4.3.33. For any two NBA $\mathcal{A}, \mathcal{B}$ over Σ, a lasso game is $\mathcal{G}_L(\mathcal{A}, \mathcal{B}) = ((V, V_0, V_1, E), v_0, \mathsf{Play})$ where DUPLICATOR's and SPOILER's configurations are $V_0 = Q \times (\Sigma^*/\approx)^3 \times Q \times \{\mathsf{D}\}$, and $V_1 \subseteq Q \times \Sigma^*/\approx \times Q \times \{\mathsf{S}\}$, $V = V_0 \cup V_1$, the edge relation E is defined as

$$(p, [w], q, \mathsf{S}) \to (p', [w], [u], [v], q, \mathsf{D}) \text{ in } E \qquad \text{iff} \qquad \begin{array}{l} p \xrightarrow{[u]} p' \xrightarrow[F]{[v]} p' \text{ and} \\ [u], [v] \text{ is proper} \end{array}$$

$$(p, [w], [u], [v], q, \mathsf{D}) \to (p, [v], q', \mathsf{S}) \text{ in } E \qquad \text{iff} \qquad q \xrightarrow{[w][u]} q' \xrightarrow[F]{[v]} q',$$

the initial configuration is $v_0 = (p_0, [\epsilon], q_0, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and the winning condition is Play, the set of all infinite plays in $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$.

Note that by the definition of the winning condition, DUPLICATOR wins any infinite play in the lasso game. Hence we can also see the lasso game $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ as a parity game where every node has an even priority. Consider a parity game $\mathcal{G}$ that is played on the configuration graph of $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ with a priority function that assigns priority 0 to all the nodes in $\mathcal{G}$. It is not hard to see that DUPLICATOR wins the lasso game $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ iff player 0 wins the parity game $\mathcal{G}$.

The size of the parity game $\mathcal{G}$ is the same as the size of the configuration graph of $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$. In the preliminary chapter, we have seen that the number of equivalence classes in $\Sigma^*/\approx$ is exponential in the size of the automata. If Q is the set of states of $\mathcal{A}$ and $\mathcal{B}$, we have $|\Sigma^*/\approx| = O(3^{Q^2})$. Hence the number of nodes and edges in the lasso game $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ is $|V| = |E| = O(3^{3Q^2})$. By Proposition 2.4.2, since we can decide whether player 0 wins such a parity game of n nodes and m edges in time $O(n{\cdot}m)$, we have the following theorem.

Theorem 4.3.34. *Deciding whether* DUPLICATOR *wins* $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ *is in* $O(3^{6(|\mathcal{A}|+|\mathcal{B}|)^2})$.

We will show that the winning strategy for DUPLICATOR in the buffered simulation game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ can be derived from the one in the lasso game $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$. First note that in every round of the lasso game $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$, SPOILER chooses a representative of a proper accepting lasso and DUPLICATOR responds to this by choosing the corresponding representative of the accepting lasso. This intuitively corresponds to the situation where in the buffered simulation game $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$, DUPLICATOR lets SPOILER play indefinitely until he forms a proper accepting lasso and repeats the cycle part. Once SPOILER does this then DUPLICATOR moves by forming a corresponding accepting lasso that she would have chosen in the lasso game. DUPLICATOR keeps playing in such a way for the rest of the play. If in the lasso game DUPLICATOR never gets stuck then DUPLICATOR can mimic any accepting lasso that is formed by SPOILER in the buffered simulation game.

Theorem 4.3.35. *If* DUPLICATOR *wins* $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ *then she also wins* $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$.

Proof. Suppose DUPLICATOR wins $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$. We will show a winning strategy for DUPLICATOR in $\mathcal{G}^\omega(\mathcal{A}, \mathcal{B})$ where she only pops the buffer and proceeds to some configuration $(p, w, q, \top, \mathsf{S})$ where w is idempotent, i.e. $[w] = [ww]$.

Suppose we are at some configuration (p, w, q, b, S) where $b \in \{\top, \bot\}$ and $w \in \Sigma^*$ is idempotent. DUPLICATOR waits until SPOILER forms a proper accepting lasso and repeats the cycle part once, i.e. until SPOILER forms

$$r = p \xrightarrow{u} p' \xrightarrow[F]{v} p' \xrightarrow[F]{v'} p'$$

where $u \approx uv$ and $v \approx v' \approx vv'$. Note that if SPOILER never forms such a run then by Lemma 4.3.27 he does not form an accepting run from p. DUPLICATOR wins $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$ by waiting forever. However if SPOILER forms such a run then we reach a configuration $(p', wuvv', q, b', \mathsf{D})$. In this case, DUPLICATOR considers what she would do in the lasso game $\mathcal{G}_L(\mathcal{A},\mathcal{B})$ if we are at some configuration $c = (p, [w], q, \mathsf{S})$ and SPOILER moves to $c' = (p', [w], [u], [v], q, \mathsf{D})$. If DUPLICATOR chooses q' and proceeds to $c'' = (p', [v], q', \mathsf{S})$ then by definition of the lasso game $\mathcal{G}_L(\mathcal{A},\mathcal{B})$, there is a path

$$q \xrightarrow{wu} q' \xrightarrow[F]{v} q'$$

in $\mathcal{B}$. In the game $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$, DUPLICATOR takes such a path and proceeds to the configuration

$$(p', v', q', \top, \mathsf{S}). \tag{4.21}$$

Now the buffer contains the word v'. Since $v' \approx v$ and v is idempotent, v' is also idempotent.

By considering this strategy either at one point DUPLICATOR skips her turn for the rest of the play or DUPLICATOR moves infinitely often. In the first case, by Lemma 4.3.27, SPOILER does not form an accepting run and in the second one, DUPLICATOR reaches the configuration of the form (4.21) infinitely often. She forms an accepting run. Hence in both cases, DUPLICATOR wins the play. She wins the game $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$. □

The reverse direction of this theorem also holds. We can derive a winning strategy for DUPLICATOR in the lasso game from the winning strategy in the buffered simulation game.

Theorem 4.3.36. DUPLICATOR *wins* $\mathcal{G}_L(\mathcal{A},\mathcal{B})$ *if she wins* $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$.

Proof. Suppose DUPLICATOR wins $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$. The winning strategy for DUPLICATOR in $\mathcal{G}_L(\mathcal{A}, \mathcal{B})$ is as follows. Suppose we are at some configuration $(p, [w], q, \mathsf{S})$. If SPOILER moves to some $(p', [w], [u], [v], q, \mathsf{D})$ then DUPLICATOR considers what she would do in the game $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$ if we are at some configuration $(p, w, q, \bot, \mathsf{S})$ and SPOILER plays an accepting run over uv^ω by reading u from p to p' followed by infinitely many accepting cycles over v from p'. If DUPLICATOR plays according to the winning strategy in $\mathcal{G}^\omega(\mathcal{A},\mathcal{B})$ then at one point she forms a run of the form

$$q \xrightarrow{wuv^*} q' \xrightarrow[F]{v^+} q'. \tag{4.22}$$

Otherwise she does not form an accepting run over wuv^ω and loses the play which contradicts that DUPLICATOR plays according to some winning strategy. Note that by the definition of the lasso game $\mathcal{G}_L(\mathcal{A},\mathcal{B})$, the pair $[u], [v]$ is proper, that is, $[uv] = [u]$ and $[vv] = [v]$. Hence any word of the form wuv^* and v^+ belongs to $[w][u]$ and $[v]$, respectively. In the lasso game $\mathcal{G}_L(\mathcal{A},\mathcal{B})$, DUPLICATOR responds to this by proceeding to the configuration $(p', [v], q', \mathsf{S})$.

Hence from any configuration $(p, [w], q, \mathsf{S})$, if SPOILER moves to some arbitrary configuration $(p', [w], [u], [v], q, \mathsf{D})$, DUPLICATOR can always respond to it. Hence either SPOILER gets stuck or the play goes on for infinitely many rounds. In both cases, DUPLICATOR wins. □

Theorem 4.3.35 and Theorem 4.3.36 show that we can polynomially reduce buffered simulation with an unbounded buffer to a lasso game. The reduction is even linear because it is just the identity mapping. By Theorem 4.3.34, since we have seen that solving the lasso game can be done in time exponentially in the size of the automata, deciding whether Duplicator wins $\mathcal{G}^{\omega}(\mathcal{A}, \mathcal{B})$ is in EXPTIME. Together with Corollary 4.3.25, we have the following result.

Corollary 4.3.37. *Deciding $\sqsubseteq^{\omega}$ is EXPTIME-complete.*

4.4 Simulation with $n \geq 1$ Unbounded Buffers

Now the only question left regarding the decidability of buffered simulation is in the case where multiple buffers are involved and some of them have an unbounded capacity. Unfortunately, in this case, buffered simulation is not only undecidable, but also highly undecidable. This even already holds in the case where we only consider buffered simulation with two buffers in which one is unbounded and the other one has a capacity 0.

We will first show that deciding the negative instance of buffered simulation, i.e. whether Spoiler wins $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$, is Σ^1_1-hard by a reduction from the recurrent octant tiling problem. We then show that deciding the positive instance is also Σ^1_1-hard by considering a more involved reduction. We combine these two results and obtain a higher undecidability degree of buffered simulation. The problem is indeed hard for the class $\mathbb{B}\Sigma^1_1$, a class that is strictly bigger than the first level of the analytical hierarchy. We will also show that deciding buffered simulation is in the second level of the analytical hierarchy, i.e. it is in Δ^1_2.

4.4.1 Σ^0_1- and Π^1_1- Hardness

First we will show that deciding buffered simulation is Σ^0_1-hard by a reduction from the octant tiling problem. Recall that the octant tiling problem asks us whether there exists a tiling for the octant of the Cartesian plane [BGG97]. Hence the width of the area that has to be tiled is not fixed. The width of the first row is one, the second row is two, etc. The octant tiling problem is clearly more complex than the corridor tiling problem. Recall that in the corridor tiling problem, the width of the area that has to be tiled is fixed. We can reduce it to the flushing variant $\mathcal{G}^{\omega}_{\mathsf{Flush}}(\mathcal{A}, \mathcal{B})$ where the role of Spoiler is to produce a tiling of width $n > 0$ and Duplicator's role is to make sure that the tiling is valid. Since the width is fixed, we can encode any possible tilings in Spoiler's automaton and the vertical and horizontal mismatches in Duplicator's automaton. However, if now we consider the octant tiling problem, it is not clear anymore how to make Duplicator detect the horizontal or vertical mismatch.

Nevertheless, we will show that we can still reduce the octant tiling problem to the buffered simulation game with a similar principle as the corridor tiling problem. We construct two NBA $\mathcal{A}$, $\mathcal{B}$ such that Spoiler's role is to produce a tiling row by row in $\mathcal{A}$ and Duplicator's role is to check whether there is a mismatch.

First we will show that we can construct two NBA $\mathcal{A}$, $\mathcal{B}$ such that Spoiler is forced to produce a tiling where the length of each row keeps growing by one, i.e. he is forced to produce a word $\#w_1\#w_2\#\ldots$ where $w_i \in T^i$. The trick is to consider a second buffer of capacity 0 that is used to store two new symbols c and $c_{\#}$. We construct the automaton $\mathcal{A}$ such that each time Spoiler reads a tile or #, and pushes it to the first buffer, he also reads c or $c_{\#}$ respectively, and pushes it to the second buffer. Since the capacity of the second buffer is 0, Duplicator has to pop c or $c_{\#}$ immediately. The automaton $\mathcal{B}$ is constructed such that each time Duplicator pops c or $c_{\#}$, she compares it with the top symbol of the

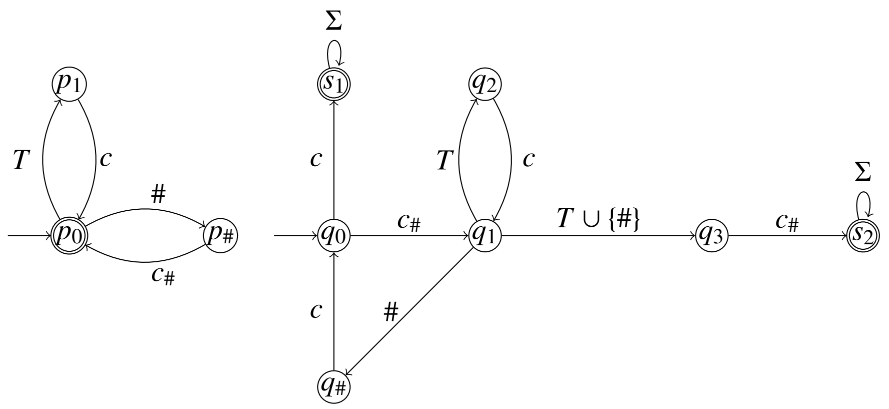


Figure 4.7: NBA $\mathcal{A}$ and $\mathcal{B}$ that force Spoiler to produce an octant tiling.

first buffer. If the top symbol is #, Duplicator pops # and then forces Spoiler to read exactly a tile and a #. However, if the top symbol is a tile then Duplicator pops it and forces Spoiler to read only a new tile. In this way, whenever the content of the first buffer is $t_1 \ldots t_n\#$, Duplicator will force Spoiler to push $t'_1, \ldots, t'_{n+1}, \#$ consecutively to the first buffer.

We illustrate such NBA $\mathcal{A}, \mathcal{B}$ in Figure 4.7. They are defined over $\hat{\Sigma} = (T \cup \{\#\}, \{c, c_\#\})$. We denote by Σ the set of all letters in $\mathcal{A}$, $\mathcal{B}$, i.e. $\Sigma = T \cup \{\#, c, c_\#\}$. The automaton $\mathcal{A}$ is very simple. It accepts any word of the form $t_1c_1t_2c_2\ldots$ where for all $i > 0$, $t_i \in T \cup \{\#\}$, and $c_i = c$ if $t_i \in T$ and $c_i = c_\#$ if $t_i = \#$. The automaton $\mathcal{B}$, however, is more involved. It basically consists of two parts. The first one that starts from q_0 and the second one from q_1. From q_0 we can reach the accepting sink s_1 by reading c, and from q_1 we can reach the accepting sink s_2 by reading $xc_\#$ where $x \in T \cup \{\#\}$. The state q_1 is reachable from q_0 by a $c_\#$-transition.

Now consider the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$. From the initial configuration $(p_0, (\epsilon, \epsilon), q_0, \bot, \mathsf{S})$, Spoiler is forced to read # since otherwise Duplicator can go to the accepting sink s_1 and win the play. Furthermore, from any configuration $(p_0, (w\#, \epsilon), q_1, \bot, \mathsf{S})$ where $w \in T^i$ and $i \geq 0$, Spoiler is forced to proceed to some configuration $(p_0, (w'\#, \epsilon), q_1, \bot, \mathsf{S})$ where $w' \in T^{i+1}$. We show this formally in the following lemma.

Lemma 4.4.1. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (T \cup \{\#\}, \{c, c_\#\})$ as in Figure 4.7 and a play in $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ from a configuration*

$$(p_0, (t_1 \ldots t_n\#, \epsilon), q_1, \bot, \mathsf{S})$$

where $t_1 \ldots t_n \in T^*$.

- Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to some configuration $(p_0, (t'_1 \ldots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$ where $t'_1, \ldots, t'_{n+1} \in T$.*
- *For any $t'_1, \ldots, t'_{n+1} \in T$,* Spoiler *has a strategy such that either* Duplicator *loses or the play proceeds to $(p_0, (t'_1 \ldots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$.*

Proof. For the first part, from the configuration $(p_0, (t_1 \ldots t_n\#, \epsilon), q_1, \bot, \mathsf{S})$, if Spoiler initially reads $\#c_\#$ then Duplicator goes to the accepting sink s_2 by popping t_1 from the first buffer and reading $t_1c_\#$. From the accepting sink, Duplicator can play accordingly and win the play. However if Spoiler reads t'_1c for some $t'_1 \in T$, Duplicator loops in q_1 by popping t_1 from the first buffer and reading t_1c. She proceeds to $(p_0, (t_2 \dots t_n\#t'_1, \epsilon), q_1, \bot, \mathsf{S})$. Duplicator then repeats this procedure for n many times. Hence either Duplicator wins by reaching the accepting sink s_2 and looping there forever or the play eventually proceeds to some configuration $(p_0, (\#t'_1 \dots t'_n, \epsilon)$, $q_1, \bot, \mathsf{S})$ where $t'_1, \dots, t'_n \in T$. Now from such a configuration, if Spoiler reads $\#c_\#$ then Duplicator goes to the accepting sink s_2 by popping # from the buffer and reading $\#c_\#$. However if Spoiler reads $t'_{n+1}c$ for some $t'_{n+1} \in T$, Duplicator pops # from the first buffer, reads $\#c$ by going to q_0 and proceeding to $(p_0, (t'_1 \dots t'_{n+1}, \epsilon)$, $q_0, \bot, \mathsf{S})$. From this configuration, if Spoiler reads tc where $t \in T$ then Duplicator goes to the accepting sink s_1 by letting t go to the first buffer and reading c. Otherwise, Spoiler reads $\#c_\#$ and in this case Duplicator proceeds to $(p_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$.

For the second part, the strategy for Spoiler is to initially read t'_1. Duplicator either skips her turn or pops t_1 from the buffer by going to q_2 or q_3. If Duplicator goes to q_3, we reach the configuration $(p_1, (t_2 \dots t_{n+1}\#t'_1, \epsilon)$, $q_3, \bot, \mathsf{S})$. In the next round, after Spoiler reads c, Duplicator gets stuck because there is no c-transition from q_3. Hence let us assume that Duplicator skips her turn or goes to q_2. Spoiler then reads c and goes back to p_0. In both cases, Duplicator has no choice except to pop c immediately and proceed to the configuration $(p_0, (t_2 \dots t_n\#t'_1, \epsilon), q_1, \bot, \mathsf{S})$. Spoiler repeats this procedure n many times to push $t'_1, \dots, t'_n$ consecutively to the buffer. Hence either Duplicator loses or the play eventually proceeds to $(p_0, (\#t'_1 \dots t'_n, \epsilon)$, $q_1, \bot, \mathsf{S})$. From such a configuration Spoiler then pushes t'_{n+1} in the same way. However in this case, after Spoiler reads t'_{n+1}, Duplicator either skips her turn or pops the top symbol of the buffer by going to $q_\#$ or q_3. If Duplicator goes to q_3, we can show similarly as before, that Duplicator eventually gets stuck and loses the play. Now assume that she skips her turn or goes to $q_\#$. Spoiler then reads c and goes back to p_0. Duplicator has no choice except to pop c immediately by going to q_0. She proceeds to $(p_0, (t'_1 \dots t'_{n+1}, \epsilon), q_0, \bot, \mathsf{S})$. Spoiler then reads $\#c_\#$ and Duplicator has no choice except to proceed to $(p_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$. □

The second part of this lemma tells us that if the first buffer contains a tiling of a row of length n then Spoiler can continue by producing a tiling of a row of length $n + 1$. The first part of the lemma then tells us that this is in fact the only way for Spoiler to continue since otherwise Duplicator can reach one of the accepting sinks and win the play.

Reduction from the Octant Tiling Problem

We can slightly extend these two NBA to reduce the octant tiling problem. We can encode the horizontal and vertical mismatches into Duplicator's automaton. Encoding the horizontal mismatch is not hard. We simply allow Duplicator to read $tt' \in T^2$ if $(t, t') \notin H$ from q_1 to the accepting sink s_2. Thus Spoiler is forced to produce a tiling that matches horizontally since otherwise Duplicator can go to s_2. Encoding the vertical mismatch, however, is a bit more involved. First recall that every time Spoiler reads a tile, he announces it by reading the new symbol c. We extend the automata such that instead of using a single symbol c for each tile $t \in T$, we use a new symbol c_t for each tile $t \in T$. If Spoiler and Duplicator are in p_0 and q_1, after Spoiler reads a tile $t \in T$ and pushes it to the first buffer, she reads c_t, instead of c, and pushes it to the second buffer. Duplicator then compares c_t with the top symbol of the first buffer. If it is #, Duplicator does the same move as before. She pops # and forces Spoiler to read exactly a tile and a #. However, if the top symbol is some $t' \in T$ then Duplicator pops it and forces Spoiler to read a new tile t that matches vertically, i.e. $(t', t) \in V$, or otherwise he goes to the accepting sink

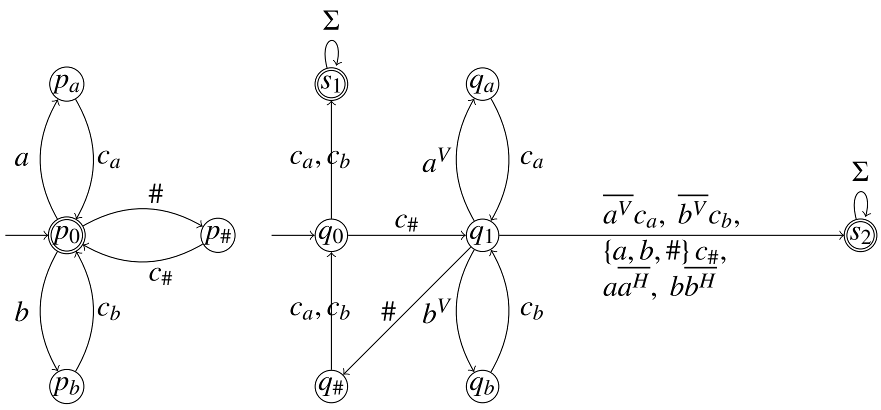


Figure 4.8: NBA $\mathcal{A}$ and $\mathcal{B}$ for the octant tiling problem.

s_2 by reading $t'\,c_t$. Hence if the content of the first buffer is $t_1 \dots t_n\#$, Spoiler is forced to push $t'_1, \dots, t'_{n+1}\#$ where $(t'_i, t'_{i+1}) \in H$ and $(t_i, t'_i) \in V$ for all $i \in \{1, \dots, n\}$.

We illustrate such extended automata in Figure 4.8. For simplicity, we assume that there are only two tiles, i.e. $T = \{a, b\}$. If there are more, we can extend the automata accordingly. Moreover, for any tile $t \in T$, we denote by t^V the set of tiles that can be put below t, i.e. $t^V = \{t' \in T \mid (t', t) \in V\}$, and by $\overline{t^V}$ the set of tiles that cannot be put below t, i.e. $\overline{t^V} = T \backslash t^V$. We also denote by $\overline{t^H}$ the set of tiles that cannot be put to the right of t, i.e. $\overline{t^H} = \{t' \in T \mid (t, t') \notin H\}$. Moreover, we abbreviate the transitions $q_1 \xrightarrow{x} \dot{q}_1 \xrightarrow{y} s_2$ with $q_1 \xrightarrow{xy} s_2$. By considering such extended automata, we can extend the property in Lemma 4.4.1 as follows.

Lemma 4.4.2. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (T \cup \{\#\}, \{c_\#, c_t \mid t \in T\})$ as in Figure 4.8 and the play in $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ from configuration*

$$(p_0, (t_1 \dots t_n\#, \epsilon), q_1, \bot, \mathsf{S})$$

where $t_1 \dots t_n \in T^*$.

- *If $(t_i, t_{i+1}) \in H$ for all $i \in \{1, \dots, n-1\}$ then for any $t'_1, \dots, t'_{n+1} \in T$ where (t_1, t'_1), …, $(t_n, t'_n) \in V$,* Spoiler *has a strategy such that either* Duplicator *loses or the play proceeds to*
$$(p_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S}).$$
- *If $(t_i, t_{i+1}) \notin H$ for some $i \in \{1, \dots, n-1\}$,* Duplicator *has a strategy to win the play.*
- *If $(t_i, t_{i+1}) \in H$ for all $i \in \{1, \dots, n-1\}$,* Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to some configuration*
$$(p_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$$
where $t'_1, \dots, t'_{n+1} \in T$ and $(t_1, t'_1), \dots, (t_n, t'_n) \in V$.

Proof. For the first part, the strategy for Spoiler is the same as in the proof of Lemma 4.4.1. Initially Spoiler reads $t'_1 c_{t'_1}$. Since $(t_1, t'_1) \in V$, Duplicator cannot go to the accepting sink s_2 by reading $t_1 c_{t'_1}$ because $t_1 \notin \overline{t'^V_1}$. Duplicator also cannot go to there by reading $t_1 t_2$ because $t_2 \notin \overline{t_1{}^H}$. Duplicator can only pop t_1 and loop in q_1 by reading $t_1 c_{t'_1}$. Hence by reading $t'_1 c_{t'_1} \dots t'_{n+1} c_{t'_{n+1}}$ and then $\#c_\#$, either Duplicator loses because she gets stuck as in the proof of Lemma 4.4.1 or the play eventually proceeds to the configuration $(p_0, (t'_1 \dots t'_{n+1}\,\#, \epsilon), q_1, \bot, \mathsf{S})$.

For the second part, let $i_0 \in \{1, \dots, n-1\}$ be the minimal index such that $(t_{i_0}, t_{i_0+1}) \notin H$. In the initial round, if Spoiler reads $\#c_\#$ or $i_0 = 1$ then Duplicator goes to the accepting sink s_2 by reading $t_1 c_\#$ or $t_1 t_2$ respectively. Otherwise, Spoiler reads $t'_1 c_{t'_1}$ for some $t'_1 \in T$ and $i_0 \neq 1$. In this case, Duplicator pops t_1 from the first buffer and reads $t_1 c_{t'_1}$ by looping in q_1. Duplicator repeats this procedure for $i_0 - 1$ many times. Hence either Duplicator eventually reaches the accepting sink s_2 and wins or the play proceeds to $(p_0, (t_{i_0} \dots t_n \# t'_1 \dots t'_{i_0-1}, \epsilon), q_1, \bot, \mathsf{S})$. From this configuration, since $(t_{i_0}, t_{i_0+1}) \notin H$, Duplicator can go to the accepting sink s_2 by reading $t_{i_0} t_{i_0+1}$ and win the play.

For the third part, Duplicator plays similarly as in the proof of Lemma 4.4.1. Additionally, from any configuration $(p_0, (t_i \dots t_n \# t'_1 \dots t'_{i-1}, \epsilon), q_1, \bot, \mathsf{S})$, if Spoiler reads $t'_i c_{t'_i}$ in which $(t_i, t'_i) \notin V$ then Duplicator goes to the accepting sink s_2 by popping t_i from the first buffer and reading $c_{t'_i}$. Hence either Spoiler loses because Duplicator reaches an accepting sink and loops there forever or the play eventually proceeds to $(p_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$ where $(t_i, t'_i) \in V$ for all $i \in \{1, \dots, n\}$. □

We can reduce the octant tiling problem to the buffered simulation game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ by considering the two NBA in Figure 4.8. If there exists an octant tiling then Spoiler simply reads the tiling row by row. Since Duplicator cannot reach any accepting sink, Spoiler will win the play. On the other hand, if there is no octant tiling then Duplicator simply makes sure that Spoiler is constructing an octant tiling row by row until he forms a vertical or horizontal mismatch. At that point, Duplicator reaches one of the accepting sinks.

Theorem 4.4.3. *Given a tiling system $\mathcal{T} = (T, H, V)$, we can construct in polynomial time two NBA $\mathcal{A}, \mathcal{B}$ such that* Spoiler *wins $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ iff there exists an octant tiling.*

Proof. Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ in Figure 4.8. Suppose there exists an octant tiling t. From the initial configuration $(p_0, (\epsilon, \epsilon), q_0, \bot, \mathsf{S})$, Spoiler first reads $\#c_\#$. Duplicator cannot do anything except to let $\#$ go to the first buffer and then pop $c_\#$ immediately from the second one. Hence we are in the configuration

$$(p_0, (\#, \epsilon), q_1, \bot, \mathsf{S}). \tag{4.23}$$

Spoiler then follows the strategy as described in the first part of Lemma 4.4.2 to push the tiling of the first row: $t_{1,1}$. Either Duplicator loses or the play proceeds to $(p_1, (t_{1,1}\#, \epsilon), q_1, \bot, \mathsf{S})$. Spoiler again follows the strategy as described in Lemma 4.4.2 to push the tiling of the second row: $t_{1,2} t_{2,2}$, and repeats the same procedure for the rest of the play. Hence either Duplicator loses or both Spoiler and Duplicator visit p_0 and q_1 infinitely often. Since Spoiler forms an accepting run and Duplicator does not, Spoiler wins.

For the other direction, suppose there is no octant tiling. Initially, if Spoiler reads $t_1\, c_{t_1}$ for some $t_1 \in T$, then Duplicator goes to the accepting sink s_1. Otherwise, Spoiler reads $\#c_\#$ and in this case, Duplicator has no choice except to proceed to the configuration (4.23). Duplicator then follows the strategy as described in the third part of Lemma 4.4.2. According to the lemma, either she wins or the play proceeds to $(p_1, (t_{1,1}\#, \epsilon), q_1, \bot, \mathsf{S})$ for some $t_{1,1} \in T$. From this configuration, Duplicator again follows the strategy as described in the third part of Lemma 4.4.2. Hence either she wins or the play proceeds to

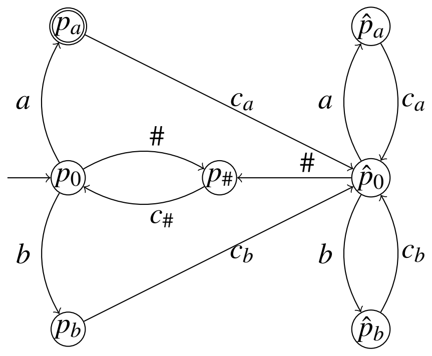


Figure 4.9: Automaton $\mathcal{A}'$ for the recurrent octant tiling problem.

$(p_1, (t_{2,1}t_{2,2}\#, \epsilon), q_1, \bot, \mathsf{S})$ for some $t_{2,1}, t_{2,2} \in T$ where $(t_{1,1}, t_{1,2}) \in V$. Now if $(t_{2,1}, t_{2,2}) \notin H$ then Duplicator follows the strategy as described in the second part of Lemma 4.4.2 to win the play. Otherwise she follows the strategy as described in the third part of Lemma 4.4.2 again. She repeats this procedure for the rest of the play. Hence either Duplicator wins or both Spoiler and Duplicator visit the states p_0 and q_1 infinitely often. In such a case, let

$$\# t_{1,1} \# t_{1,2}t_{2,2} \# t_{1,3}t_{2,3}t_{3,3} \# t_{1,4}t_{2,4}t_{3,4}t_{4,4} \# \dots \tag{4.24}$$

be the sequence of letters that are pushed by Spoiler to the first buffer. By Lemma 4.4.2, we have $(t_{i,j}, t_{i,j+1}) \in V$ and $(t_{i,j}, t_{i+1,j}) \in H$ for all $i \leq j$. Hence we can construct an octant tiling t where $t(i, j) = t_{i,j}$. This however contradicts our initial assumption. Hence Duplicator wins the play. □

We have a polynomial-time reduction from the octant tiling problem to the problem of deciding whether Duplicator wins buffered simulation game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$. Thus we have the following corollary.

Corollary 4.4.4. *Deciding $\sqsubseteq^{\omega,0}$ is Σ^0_1-hard.*

We can lift the reduction in Theorem 4.4.3 to reduce a highly undecidable problem, namely the recurrent octant tiling problem [Har85]. We encode the situation where the final tile is used as the first tile of a row infinitely often by slightly modifying the automaton for Spoiler. Instead of the automaton $\mathcal{A}$ as in Theorem 4.4.3, we consider a new automaton $\mathcal{A}'$ that simulates $\mathcal{A}$ and remembers the tile that is used as the first tile of a row in its structure. The automaton $\mathcal{A}'$ is obtained from $\mathcal{A}$ by adding a copy of it. We construct $\mathcal{A}'$ such that Spoiler uses the original part to push the first tile of a row and the copy part to push the rest of the tiles. Instead of the initial state p_0, we make the state that corresponds to the final tile t_F, i.e. p_{t_F}, in the original part, to be accepting, and the rest of the states are not. In this way, Spoiler produces an accepting run iff the final tile is used as the first tile infinitely often. We illustrate such an automaton $\mathcal{A}'$ in Figure 4.9. We assume that a is the final tile, and hence the state p_a is accepting and the rest of the states are not.

Theorem 4.4.5. *Given a tiling system $\mathcal{T} = (T, H, V)$ and a final tile $t_F \in T$, we can construct two NBA $\mathcal{A}', \mathcal{B}$ such that* Spoiler *wins $\mathcal{G}^{\omega,0}(\mathcal{A}', \mathcal{B})$ iff there exists a recurrent octant tiling.*

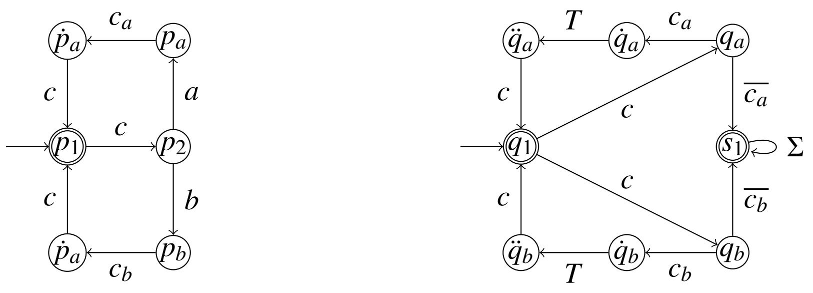


Figure 4.10: F-structure.

Proof. Consider the two automata $\mathcal{A}'$, $\mathcal{B}$, in which $\mathcal{A}'$ is the automaton that is obtained from the automaton $\mathcal{A}$ from Theorem 4.4.3 by adding a copy of it as we have illustrated in Figure 4.9 and $\mathcal{B}$ is the automaton as in Theorem 4.4.3. Suppose there exists a recurrent octant tiling t. The winning strategy for Spoiler is the same as the one in Theorem 4.4.3. He pushes $\#t_{1,1}\#t_{1,2}t_{2,2}\#\ldots$ to the first buffer. Since there exist infinitely many i such that $t_{1,i} = t_F$, he visits the accepting state q_{t_F} infinitely often and hence forms an accepting run. Since Duplicator does not form an accepting run, Spoiler wins.

For the reverse direction, suppose there is no recurrent octant tiling. The winning strategy for Duplicator is also the same as the one in the proof of Theorem 4.4.3. Hence either Duplicator wins or both Spoiler and Duplicator respectively visit p_0 and q_1 infinitely often. In such a case, Spoiler pushes a sequence of letters as in (4.24) that obeys the vertical and horizontal compatibility relation, i.e. for all $i \leq j$, $(t_{i,j}, t_{i+1,j}) \in H$ and $(t_{i,j}, t_{i,j+1}) \in V$. Now suppose there are infinitely many j such that $t_{1,j} = t_F$. In this case, we can construct a recurrent octant tiling t where $t(i, j) = t_{i,j}$. This, however, contradicts our initial assumption. Hence there are only finitely many j such that $t_{1,j} = t_F$. Spoiler visits q_{t_F} finitely often and hence does not form an accepting run. Duplicator wins the play. □

Since we can polynomially reduce the recurrent octant tiling problem to the problem of deciding $\not\sqsubseteq^{\omega,0}$, we have the following corollary.

Corollary 4.4.6. *Deciding* $\sqsubseteq^{\omega,0}$ *is* Π^1_1*-hard.*

4.4.2 Π^0_1- and Σ^1_1- Hardness

Corollary 4.4.6 shows that we can reduce the recurrent octant tiling problems to the negative instance of buffered simulation. One then can equally ask whether the problem can also be reduced to the positive instance of buffered simulation in a similar way. It is however not clear how to do so since we need to switch the role of the players. We have to make Duplicator as the one that produces the tiling, and not Spoiler. This seems to be against the nature of buffered simulation since the player that has the role to produce is Spoiler, whereas Duplicator's role is simply to mimic what Spoiler has produced. Nevertheless, we will show that it is possible to make Duplicator the player that chooses the tiling. The trick is to use the following special structure.

F-structure

Given a tiling system $\mathcal{T} = (T, H, V)$, consider the two NBA $\mathcal{A}_\mathsf{F}$, $\mathcal{B}_\mathsf{F}$ as in Figure 4.10 over the distributed alphabet $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ where $\Sigma_1 = T$ and $\Sigma_2 = \{c\} \cup \{c_t \mid t \in T\}$. For simplicity, we again assume that the set of tiles T consists only two tiles a and b. If there are more, the automata can be extended accordingly. For any tile $t \in T$, we denote by $\overline{c_t}$ the set of letters in Σ_2 that are not c_t, i.e. $\overline{c_t} = \Sigma_2 \backslash \{c_t\}$. We also denote by Σ the set of all letters in $\mathcal{A}$, $\mathcal{B}$, i.e. $\Sigma = \Sigma_1 \cup \Sigma_2$.

Consider the game $\mathcal{G}^{\omega,0}(\mathcal{A}_\mathsf{F}, \mathcal{B}_\mathsf{F})$ and a play from the initial configuration. Duplicator can choose a tile $t \in T$ and force Spoiler to push it to the first buffer. The strategy is simply to read $cc_t tc$. Spoiler will push t to the first buffer since otherwise Duplicator can reach the accepting sink s_1. We generalise such a property to the following lemma that will be the key of our reduction.

Lemma 4.4.7. *Consider the two NBA $\mathcal{A}_\mathsf{F}$, $\mathcal{B}_\mathsf{F}$ over $\hat{\Sigma} = (T, \{c\} \cup \{c_t \mid t \in T\})$ as in Figure 4.7 and a play in $\mathcal{G}^{\omega,0}(\mathcal{A}_\mathsf{F}, \mathcal{B}_\mathsf{F})$ from some configuration*

$$(p_1, (t_1 \dots t_n, \epsilon), q_1, x, \mathsf{S}) \tag{4.25}$$

where $t_1, \dots, t_n \in T$, $n > 0$, and $x \in \{2, \perp, \top\}$.

- *For any $t \in T$,* Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to*

  $$(p_1, (t_2 \dots t_n\, t, \epsilon), q_1, x', \mathsf{S})$$

  where $x' = \top$ if $x \in \{2, \perp\}$ and $x' = 2$ if $x = \top$.

- Spoiler *has a strategy such that the play proceeds to some configuration*

  $$(p_1, (t_2 \dots t_n\, t, \epsilon), q_1, x', \mathsf{S})$$

  where $t \in T$, $x' = \top$ if $x \in \{2, \perp\}$ and $x' \in \{2, \perp\}$ if $x = \top$.

Proof. For the first part, the strategy for Duplicator is as follows. From the configuration (4.25), after Spoiler reads c, Duplicator pops c by going to q_t, i.e. she proceeds to $(p_2, (t_1 \dots t_n, \epsilon), q_t, x', \mathsf{S})$ where by definition of $\mathcal{G}^{\omega,0}(\mathcal{A}_\mathsf{F}, \mathcal{B}_\mathsf{F})$, $x' = \perp$ if $x \in \{2, \perp\}$ and $x' = 1$ if $x = \top$. Spoiler then will read some tile $t' \in T$ by going to $p_{t'}$ and Duplicator cannot do anything except to skip her turn and proceed to $(p_{t'}, (t_1 \dots t_n\, t', \epsilon), q_t, x', \mathsf{S})$. From this configuration, Spoiler will read $c_{t'}$. If $t' \neq t$, Duplicator goes to the accepting s_1 by reading $c_{t'} \in \overline{c_t}$. From the accepting sink, Duplicator can continue accordingly and win the play. However, if $t' = t$ then Duplicator goes to $\dot{q}_t$ by popping c_t from the buffer. We reach the configuration $(\dot{p}_t, (t_1 \dots t_n\, t, \epsilon), \dot{q}_t, x', \mathsf{S})$. Spoiler then will read c and Duplicator responds to this by going to q_1, i.e. she proceeds to $(p_1, (t_2 \dots t_n\, t, \epsilon), q_1, x'', \mathsf{S})$ where by definition of $\mathcal{G}^{\omega,0}(\mathcal{A}_\mathsf{F}, \mathcal{B}_\mathsf{F})$, x'' is $\top$ if $x' = \perp$ and 2 if $x' = 1$.

For the second part, the strategy for Spoiler is as follows. First he reads c. If Duplicator responds to this by going to q_t then Spoiler reads t by going to p_t. Duplicator will skip her turn and we reach the configuration $(p_t, (t_1 \dots t_n\, t, \epsilon), q_t, x', \mathsf{S})$ where $x' = \perp$ if $x \in \{2, \perp\}$ and $x' = 1$ if $x = \top$. Spoiler then reads c_t. Duplicator cannot go to the accepting sink since $c_t \notin \overline{c_t}$. However, she can read c_t by going to $\dot{q}_t$ or read $c_t\, t_1$ by going to $\ddot{q}_t$. Hence the play respectively proceeds to the configuration

$$(\dot{p}_t, (t_1 \dots t_n\, t, \epsilon), \dot{q}_t, x', \mathsf{S}) \text{ or} \tag{4.26}$$

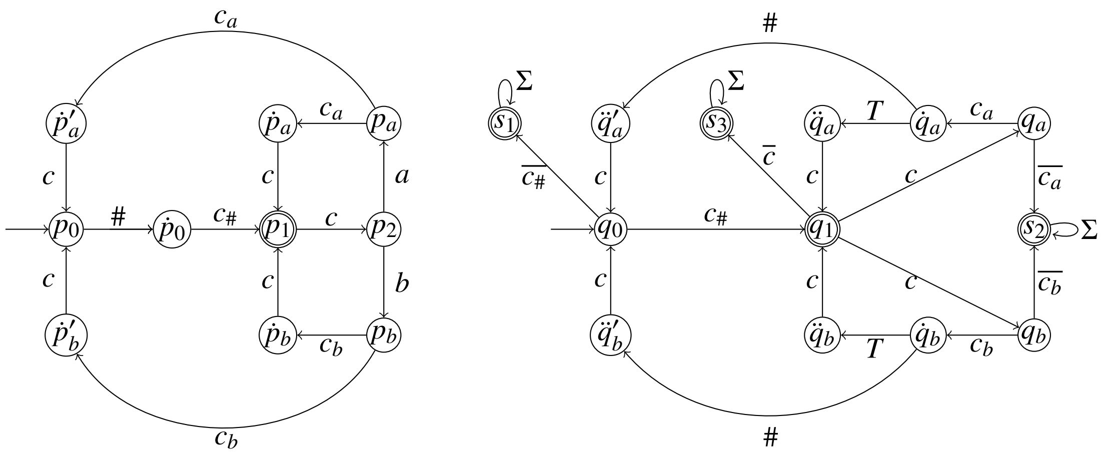


Figure 4.11: NBA $\mathcal{A}$ and $\mathcal{B}$ to force Duplicator to produce an octant tiling.

$$(\dot{p}_t, (t_2 \dots t_n\, t, \epsilon), \ddot{q}_t, x'', \mathsf{S}) \tag{4.27}$$

where $x'' = \bot$ if $x' = \bot$ and $x'' = 2$ if $x' = 1$. In the next round, Spoiler will read c and Duplicator does not have any choice except to go to p_1 and proceed to some configuration $(p_1, (t_2 \dots t_n\, t, \epsilon), q_1, x''', \mathsf{S})$ where $x''' \in \{2, \bot, \top\}$. If she comes from (4.26) then x''' is $\top$ if $x' = \bot$, and 2 if $x' = 1$. However if she comes from (4.27) then x''' is $\top$ if $x'' = \bot$, and $\bot$ if $x'' = 2$. □

The first part of this lemma shows that Duplicator can choose any tile and force Spoiler to push it to the first buffer. The second part then shows that this is the only thing that Duplicator can do. She cannot reach the accepting sink if Spoiler reads the tile of Duplicator's choice accordingly.

We will use such an F-structure to enable Duplicator to choose a tiling of a row and force Spoiler to push the tiles one by one to the first buffer.

Reduction from the Octant Tiling Problem

For a given tiling system $\mathcal{T}$, first we will show that we can construct two NBA $\mathcal{A}$, $\mathcal{B}$ such that Duplicator has to choose the words $w_1, w_2, \dots$ where $w_i \in T^i$ and force Spoiler to push $\#w_1\#w_2\dots$ to the buffer. The principle is similar to the automata given in Lemma 4.4.1, but we use the F-structure to make Spoiler reads the tiles that Duplicator chooses. Initially, Spoiler is forced to push # and then a tile t_1 of Duplicator's choice. Each time Spoiler pushes a tile t_i, $i > 0$, Duplicator pops the top symbol of the first buffer. If the top symbol is a tile, Spoiler is forced to push only the tile t_i. However if it is #, Spoiler is forced to push # after he pushes the tile t_i. Duplicator then continues to force Spoiler to push some tile t_{i+1} of her choice. Thus if the content of the first buffer is $t_1 \dots t_n\#$, Spoiler then is forced to push $t'_1, \dots, t'_{n+1}$, # consecutively to the buffer.

We illustrate such NBA $\mathcal{A}$, $\mathcal{B}$ in Figure 4.11. They are defined over the distributed alphabet $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ where $\Sigma_1 = T \cup \{\#\}$ and $\Sigma_2 = \{c, c_\#\} \cup \{c_t \mid t \in T\}$. Each of the automata $\mathcal{A}$ and $\mathcal{B}$ intuitively consists of two parts. The left parts that start from p_0 and q_0, and the right parts that start from p_1 and q_1. Note that the right parts are nonetheless the F-structure.

Consider the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$. From the initial configuration $(p_0, (\epsilon, \epsilon), q_0, \bot, \mathsf{S})$, Spoiler will read # and then $c_\#$. The play then proceeds to the configuration $(p_1, (\#, \epsilon), q_1,$

$\top, \mathsf{S})$. Furthermore, from any configuration $(p_1, (w\#, \epsilon), q_1, \top, \mathsf{S})$ where $w \in T^i$ and $i \geq 0$, by Lemma 4.4.7, we can show that Duplicator can choose $w' \in T^{i+1}$ and force Spoiler to push the letters of w' one by one to the first buffer and proceed to the configuration $(p_1, (w'\#, \epsilon), q_1, \top, \mathsf{S})$. We show this formally in the following lemma.

Lemma 4.4.8. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (T \cup \{\#\}, \{c, c_\#\} \cup \{c_t \mid t \in T\})$ as in Figure 4.11 and a play in $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ from some configuration*

$$(p_1, (t_1 \dots t_n\#, \epsilon), q_1, \top, \mathsf{S}) \tag{4.28}$$

where $t_1 \dots t_n \in T^*$.

- *For any $t'_1, \dots, t'_{n+1} \in T$,* Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to some configuration*

$$(p_1, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S}).$$

- Spoiler *has a strategy such that the play eventually proceeds to some configuration*

$$(p_1, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$$

where $t'_1, \dots, t'_{n+1} \in T$.

Proof. For the first part, from the configuration (4.28), Duplicator follows the strategy as described in the first part of Lemma 4.4.7 to push the tile t'_1. Hence either Duplicator wins or the play proceeds to $(p, (t_2 \dots t_n\#t'_1, \epsilon), q_1, 2, \mathsf{S})$ where $p \in \{p_0, p_1\}$. If $p = p_0$, Spoiler then will read # and then $c_\#$. In such a case, Duplicator first skips her turn and then goes to the accepting sink s_3 by reading $c_\# \in \overline{c}$. Duplicator can play accordingly from the accepting sink and win the play. However if $p = p_1$, Duplicator again follows the strategy from the first part of Lemma 4.4.7 to push t'_2. She repeats this procedure n many times to push $t'_1, \dots, t'_n$ to the buffer. According to Lemma 4.4.7, either Duplicator wins or the play eventually proceeds to $(p_1, (\#t'_1 \dots t'_n, \epsilon), q_1, x, \mathsf{S})$ where $x \in \{2, \top\}$. From such a configuration, Duplicator again follows the same strategy to force Spoiler to push t'_{n+1} to the buffer. In this case, either Duplicator wins or the play proceeds to $(p, (t'_1 \dots t'_{n+1}, \epsilon), q_0, x', \mathsf{S})$ where $p \in \{p_0, p_1\}$. If $p = p_1$ then Spoiler will continue by reading c. Duplicator goes to the accepting sink s_1 by reading $c \in \overline{c_\#}$ and plays accordingly from the accepting sink to win the play. However if $p = p_0$ then we reach the configuration $(p_0, (t'_1 \dots t'_{n+1}, \epsilon), q_0, x', \mathsf{S})$ where $x' \in \{2, \bot\}$. Spoiler then will read #. Duplicator skips her turn, and hence proceeds to the configuration $(\dot{p}_0, (t'_1 \dots t'_{n+1}\#, \epsilon), q_0, \bot, \mathsf{S})$. Spoiler will continue by reading $c_\#$ and Duplicator responds to this by popping $c_\#$ immediately. The play then proceeds to $(p_1, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$.

For the second part, Spoiler first follows the strategy as described in the proof of Lemma 4.4.7. He lets Duplicator force him to push some $t'_1 \in T$ and then goes back to p_1. Hence we reach the configuration $(p_1, (t_2 \dots t_n\#t'_1, \epsilon), q_1, x, \mathsf{S})$ where $x \in \{2, \bot\}$. Spoiler repeats this procedure for n many times. By Lemma 4.4.7, the play eventually proceeds to $(p_2, (\#t'_1 \dots t'_n, \epsilon), q_1, x', \mathsf{S})$ where $t'_1, \dots, t'_n \in T$ and $x' \in \{2, \bot, \top\}$. From such a configuration, Spoiler again lets Duplicator force him to push some $t'_{n+1} \in T$. However now he goes to q_0. Hence we reach some configuration $(p_0, (t'_1 \dots t'_{n+1}, \epsilon), q_0, x', \mathsf{S})$ where $x' = \bot$ if $x \in \{\bot, 2\}$ and $x' \in \{2, \bot\}$ if $x = \top$. Spoiler reads # and then $c_\#$. Duplicator first skips her turn and proceeds to $(p_1, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \bot, \mathsf{S})$. She then pops $c_\#$ and proceeds to $(p_1, (t'_1 \dots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$. □

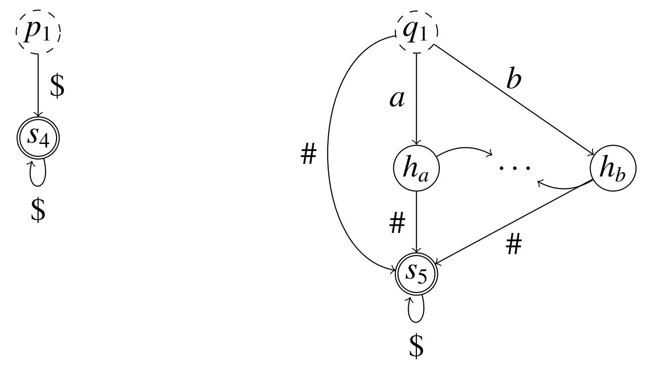


Figure 4.12: The components $\mathcal{A}_{\mathsf{check}}$ and $\mathcal{B}_{\mathsf{check}}$ to ensure horizontal compatibility.

The first part of this lemma tells us that if the first buffer contains a tiling of a row of length n then Duplicator can choose a tiling of a row of length $n+1$ and force Spoiler to push the tiles one by one to the buffer. The second part tells us that this is in fact the only way for Duplicator to continue.

We can slightly extend these two automata to reduce the octant tiling problem. We encode the horizontal mismatch by adding two new components $\mathcal{A}_{\mathsf{check}}$ and $\mathcal{B}_{\mathsf{check}}$ as illustrated in Figure 4.12. The new components $\mathcal{A}_{\mathsf{check}}$ and $\mathcal{B}_{\mathsf{check}}$ are defined over $T \cup \{\#, \$\}$ where $ is a new symbol that belongs to the first buffer. The component $\mathcal{A}_{\mathsf{check}}$ is very simple. It accepts the word $\$^\omega$ from the state p_1. The component $\mathcal{B}_{\mathsf{check}}$ is more involved. It consists of an accepting sink and a state h_t for each tile $t \in T$. The accepting sink is reachable from q_1 by reading # and from each state h_t by reading t. Moreover for all $t \in T$, we have $h_t \xrightarrow{t'} h_{t'}$ iff $(t, t') \in H$. Hence from q_1, the component $\mathcal{B}_{\mathsf{check}}$ accepts any word of the form $t_1 \dots t_m \# \$^\omega$ where $m > 0$ and $(t_i, t_{i+1}) \in H$ for all $i \in \{1, \dots, m-1\}$.

Now if we extend the automata $\mathcal{A}$, $\mathcal{B}$ with $\mathcal{A}_{\mathsf{check}}$, $\mathcal{B}_{\mathsf{check}}$ then from any configuration $(p_1, (t_1 \dots t_n\#, \epsilon), q_1, \top, \mathsf{S})$, where $t_1, \dots, t_n \in T$, if there is some $i \in \{1, \dots, n\}$ such that $(t_i, t_{i+1}) \notin H$, Spoiler wins by going to the accepting sink s_4 in $\mathcal{A}_{\mathsf{check}}$ and loops there forever. Duplicator will never reach the accepting sink s_5 in $\mathcal{B}_{\mathsf{check}}$ and lose the play. Spoiler, however, should not go to s_4 if there is no such i since Duplicator can reach s_5 and win the play. Hence Spoiler should only use the new component $\mathcal{A}_{\mathsf{check}}$ if there is a horizontal mismatch.

Encoding the vertical mismatch to the automata $\mathcal{A}$, $\mathcal{B}$ is also simple. First recall that each time after Duplicator forces Spoiler to read a new tile, she pops a tile or # from the top of the first buffer. The tile that Duplicator pops is indeed the one that is put below the new tile. Up to now, Duplicator can pop any tile. Hence to encode the vertical compatibility, we simply restrict the tile that Duplicator can pop. It should only be the one that matches vertically with the new tile. Hence from any state $\dot{q}_t$, Duplicator can only reach $\ddot{q}_t$ by reading t' where $(t', t) \in V$. Let us denote with t^V the set of tiles that can be put below t, i.e. $t^V = \{t' \mid (t', t) \in V\}$. We illustrate the extended two NBA $\mathcal{A}$, $\mathcal{B}$ that encode the horizontal and vertical mismatch in Figure 4.12. By considering such automata, we can extend the property in Lemma 4.4.8 as follows.

Lemma 4.4.9. *Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (T \cup \{\#, \$\}, \{c, c_\#\} \cup \{c_t \mid t \in T\})$ as in Figure 4.13 and a play in $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ from*

$$(p_1, (t_1 \dots t_n\#, \epsilon), q_1, \top, \mathsf{S})$$

where $t_1 \dots t_n \in T^*$.

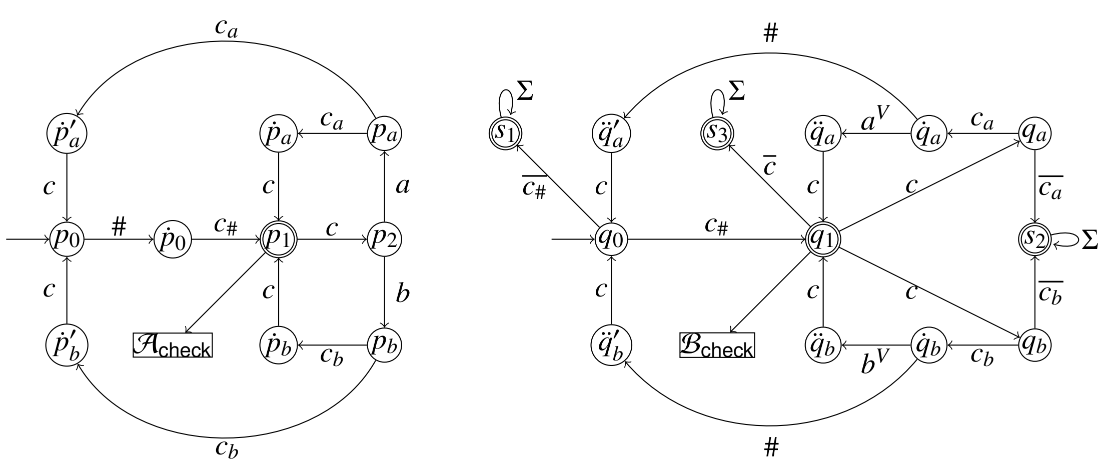


Figure 4.13: NBA $\mathcal{A}$, $\mathcal{B}$ for the octant tiling problem.

- *If* $(t_i, t_{i+1}) \in H$ *for all* $i \in \{1, \ldots, n-1\}$ *then for any* $t'_1, \ldots, t'_{n+1} \in T$ *where* (t_1, t'_1), *...,* $(t_1, t'_1) \in V$, Duplicator *has a strategy such that either* Spoiler *loses or the play proceeds to* $(p_1, (t'_1 \ldots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$.

- *If* $(t_i, t_{i+1}) \notin H$ *for some* $i \in \{1, \ldots, n-1\}$ *then* Spoiler *has a strategy to win the play.*

- *If* $(t_i, t_{i+1}) \in H$ *for all* $i \in \{1, \ldots, n-1\}$ *then* Spoiler *has a strategy such that either* Duplicator *loses or the play proceeds to* $(p_1, (t'_1 \ldots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$ *where* $(t_i, t'_i) \in V$ *for all* $i \in \{1, \ldots, n\}$.

Proof. For the first part, the strategy for Duplicator is as follows. If initially Spoiler goes to the accepting sink s_4 in $\mathcal{A}_{\text{check}}$, Duplicator then goes to the accepting sink s_5 in $\mathcal{B}_{\text{check}}$ by reading $t_1 \ldots t_n\#$. This is possible because $(t_i, t_{i+1}) \in H$ for all $i \in \{1, \ldots, n-1\}$. From the accepting sink, Duplicator can play accordingly and win the play. However, if Spoiler does not go to the accepting sink s_4 then Duplicator follows the strategy as described in the first part of Lemma 4.4.8.

For the second part, the strategy for Spoiler is simply to go to the accepting sink s_4 in $\mathcal{A}_{\text{check}}$, i.e. he proceeds to $(s_4, (t_1 \ldots t_n\#\$, \epsilon), q_1, \top, \mathsf{D})$, and then loops in s_4 forever. Since $(t_i, t_{i+1}) \notin H$ for some $i \in \{1, \ldots, n-1\}$, Duplicator will never be able to pop \$ from the buffer. Spoiler wins by looping in s_4 forever.

For the third part, Spoiler follows the strategy as described in the proof of Lemma 4.4.8. Note that from any configuration $(p_1, (t_i \ldots t_n\#t'_1 \ldots t'_{i-1}, \epsilon), q_1, x, \mathsf{S})$ where $x \in \{2, \bot, \top\}$, Spoiler reads c. Duplicator will read c by going to some $q_{t'_i}$ and proceeding to some configuration $(p_2, (t_i \ldots t_n\ \#t'_1 \ldots t'_{i-1}, \epsilon), q_{t'_i}, x', \mathsf{S})$ where $x \in \{\bot, 1\}$. From this configuration, Spoiler reads t'_i and then $c_{t'_i}$. If $(t_i, t'_i) \notin V$, since there is no t_i-transition from $\dot{q}_{t'_i}$, Duplicator will get stuck and lose the play. Otherwise the play proceeds as in the proof of Lemma 4.4.8, to some configuration $(p_1, (t_{i+1} \ldots t_n\ \#\ t'_1 \ldots t'_i, \epsilon), q_1, x'', \mathsf{S})$ where $x'' \in \{2, \bot, \top\}$. Hence either Duplicator loses or the play eventually proceeds to some configuration $(p_1, (t'_1 \ldots t'_{n+1}\#, \epsilon), q_1, \top, \mathsf{S})$ where $(t_i, t'_i) \in V$ for all $i \in \{1, \ldots, n\}$. □

We can reduce the octant tiling problem to the buffered simulation game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ by considering such NBA $\mathcal{A}$, $\mathcal{B}$. If there exists an octant tiling then Duplicator simply forces Spoiler to read the tiling row by row. If Spoiler does not obey, Duplicator can reach one of the accepting sinks and win the play, otherwise she goes through the accepting state infinitely often. On the other hand, if there is no octant tiling, Spoiler simply reads every tile that is chosen by Duplicator until there is a vertical or horizontal mismatch. In the first case, Duplicator eventually gets stuck and loses the play, and in the second one, Spoiler can go to the accepting sink in $\mathcal{A}_{\mathsf{check}}$ and win the play. We show this formally in the following theorem.

Theorem 4.4.10. *Given a tiling system $\mathcal{T} = (T, H, V)$, we can construct in polynomial time two NBA $\mathcal{A}, \mathcal{B}$ such that* Duplicator *wins $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$ iff there exists an octant tiling.*

Proof. Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ that are illustrated in Figure 4.13. Suppose there exists an octant tiling t. In the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$, initially Spoiler will read # then $c_\#$, and Duplicator has no choice except to proceed to

$$(p_1, (\#, \epsilon), q_1, \top, \mathsf{S}). \tag{4.29}$$

From this configuration, Duplicator follows the strategy as described in Lemma 4.4.9. She forces Spoiler to read the tiling of the first row: $t_{1,1}$. According to the lemma, either Duplicator wins or the play proceeds to $(p_1, (t_{1,1}\#, \epsilon), q_1, \top, \mathsf{S})$. Duplicator then follows the strategy as described in Lemma 4.4.9 again. She forces Spoiler to read the tiling of the second row: $t_{1,2}t_{2,2}$. She repeats this procedure for the rest of the play. Hence either Duplicator wins by following the strategy as described in Lemma 4.4.9 or both Spoiler and Duplicator go through the states p_1 and q_1 infinitely often. In the second case, Duplicator also wins the play.

For the other direction, suppose there is no octant tiling. Initially, Spoiler reads $\#c_\#$. The play then proceeds to (4.29). From such a configuration, Spoiler follows the strategy as in the third part of Lemma 4.4.9. Hence either Duplicator loses or the play proceeds to $(p_1, (t_{1,1}\#, \epsilon), q_1, \top, \mathsf{S})$ for some $t_{1,1} \in T$. From this configuration, Spoiler again follows the strategy as described in the third part of Lemma 4.4.9. Either Duplicator loses or the play proceeds to $(p_1, (t_{2,1}t_{2,2}\#, \epsilon), q_1, \top, \mathsf{S})$ for some $t_{2,1}, t_{2,2} \in T$. Now if $(t_{2,1}, t_{2,2}) \notin H$, Spoiler follows the strategy as in the second part of Lemma 4.4.9 to win the play. Otherwise, he follows the strategy as described in the third part again. Spoiler repeats this procedure indefinitely. Hence either he wins by following the strategy as described in Lemma 4.4.9 or both Spoiler and Duplicator go through p_1 and q_1 infinitely often. In such a case, let

$$\#\, t_{1,1}\, \#\, t_{1,2}t_{2,2}\, \#\, t_{1,3}t_{2,3}t_{3,3}\, \#\, t_{1,4}t_{2,4}t_{3,4}t_{4,4}\, \# \ldots$$

be the sequence of letters that are pushed by Spoiler to the first buffer. By Lemma 4.4.9, $(t_{i,j}, t_{i,j+1}) \in V$ and $(t_{i,j}, t_{i+1,j}) \in H$ for all $i \leq j$. Hence we can construct an octant tiling t where $t(i, j) = t_{i,j}$. This however contradicts our initial assumption. Spoiler wins the play. □

We have a polynomial-time reduction from the octant tiling problem to the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$. Thus we have the following corollary.

Corollary 4.4.11. *Deciding $\sqsubseteq^{\omega,0}$ is Π^0_1-hard.*

We can also lift the reduction in Theorem 4.4.10 to reduce the recurrent octant tiling problem. Instead of the automaton $\mathcal{B}$, we will consider a new automaton $\mathcal{B}'$ for Duplicator, that simulates $\mathcal{B}$ and remembers the tile that is used as the first tile of a row in its structure. The automaton $\mathcal{B}'$ is obtained from $\mathcal{B}$ by simply adding a copy of the F-structure of $\mathcal{B}$. We construct $\mathcal{B}'$ such that Duplicator uses the original F-structure to

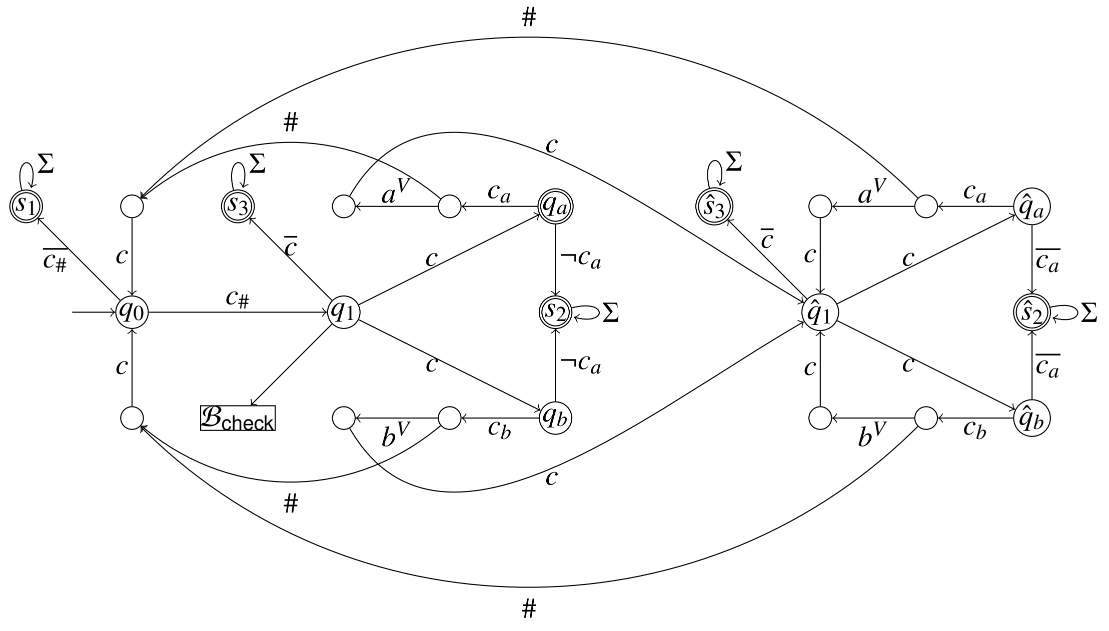


Figure 4.14: Automaton $\mathcal{B}''$ for the recurrent octant tiling problem.

force Spoiler to read the first tile of a row, and then uses the copy for the rest of the tiles. Instead of the state q_1, we make the state in the original F-structure that corresponds to the final tile t_F, i.e. q_{t_F}, to be accepting, and the rest of the non-sink nodes non-accepting. In this way, Duplicator forms an accepting run iff the final tile is used as the first tile of a row infinitely often. We illustrate such an extended automaton $\mathcal{B}'$ in Figure 4.14. We assume that a is the final tile, and hence the state q_a is accepting and the rest of the non-sink states are not.

Theorem 4.4.12. *Given a tiling system $\mathcal{T} = (T, H, V)$ and a final tile $t_F \in T$, we can construct two NBA $\mathcal{A}, \mathcal{B}'$ such that* Duplicator *wins $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B}')$ iff there exists a recurrent octant tiling.*

Proof. Consider two NBA $\mathcal{A}, \mathcal{B}'$ where $\mathcal{A}$ is the automaton as in Theorem 4.4.10 and $\mathcal{B}'$ is obtained from the automaton $\mathcal{B}$ from Theorem 4.4.10 by adding a copy of the F-structure as we have illustrated in Figure 4.14. Suppose there exists a recurrent octant tiling t. The winning strategy for Duplicator is intuitively the same as the one in Theorem 4.4.10. She forces Spoiler to push $\#t_{1,1}\#t_{1,2}t_{2,2}\#\ldots$ to the first buffer. Since there exist infinitely many i such that $t_{1,i} = t_F$, Duplicator visits the accepting state q_{t_F} infinitely often and hence wins the play.

For the reverse direction, suppose there is no recurrent octant tiling. The winning strategy for Spoiler is also the same as the one in Theorem 4.4.10. She simply reads every letter that is forced by Duplicator and reaches the accepting sink in $\mathcal{A}_{\text{check}}$ if there is a horizontal mismatch. Hence either Spoiler wins because Duplicator produces a vertical or horizontal mismatch, or both Spoiler and Duplicator visit the states p_1 and q_1 infinitely often. In the second case, let

$$\#\, t_{1,1}\, \#\, t_{1,2}t_{2,2}\, \#\, t_{1,3}t_{2,3}t_{3,3}\, \#\, t_{1,4}t_{2,4}t_{3,4}t_{4,4}\, \#\ldots$$

be the sequence of letters that are pushed to the first buffer. Since there is no vertical or horizontal mismatch, we have $(t_{i,j}, t_{i+1,j}) \in H$ and $(t_{i,j}, t_{i,j+1}) \in V$ for all $i \leq j$. Now suppose there are infinitely many j such that $t_{1,j} = t_F$. In this case, we can construct a recurrent octant tiling t where $t(i, j) = t_{i,j}$. This however contradicts our initial assumption. Hence there are only finitely many j such that $t_{1,j} = t_F$. Duplicator visits q_{t_F} finitely often and hence does not form an accepting run. Spoiler, on the other hand, forms an accepting run. Thus Spoiler wins. □

Since we can polynomially reduce the recurrent octant tiling problem to the problem of deciding whether Duplicator wins buffered simulation game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$, we have the following corollary.

Corollary 4.4.13. *Deciding $\sqsubseteq^{\omega,0}$ is Σ^1_1-hard.*

In the next subsection, we will show that we can use Corollary 4.4.11 and Corollary 4.4.13 to show that solving buffered simulation is not only hard for the first level of the analytical hierarchy, but also hard for a higher class $\mathbb{B}\Sigma^1_1$ that strictly contains Σ^1_1 and Π^1_1.

4.4.3 $\mathbb{B}\Sigma^1_1$-Hardness

Before we show the $\mathbb{B}\Sigma^1_1$-hardness of buffered simulation, let us shortly recall a decision problem that asks whether a property $P(x_1, \dots, x_n)$ holds for given $x_1, \dots, x_n \in \mathbb{N}$, P. Since the outcome of P is either positive or negative, we can also see the boolean combination of decision problems as a decision problem. Intuitively, for a problem P, $\neg P$ is a problem that asks whether the outcome of P is not positive, $P_1 \wedge P_2$ asks whether the outcome of both P_1 and P_2 are positive, and $P_1 \vee P_2$ asks whether the outcome of one of P_1 and P_2 is positive.

Let us denote by $\mathbb{B}\Sigma^1_1$ the set of all boolean combinations of problems in Σ^1_1, i.e. a problem P is in $\mathbb{B}\Sigma^1_1$ if either P is in Σ^1_1, P is $\neg P'$ and P' is in $\mathbb{B}\Sigma^1_1$, P is $P_1 \wedge P_2$ and P_1, P_2 are in $\mathbb{B}\Sigma^1_1$, or P is $P_1 \vee P_2$ and P_1, P_2 are in $\mathbb{B}\Sigma^1_1$. Note that the class $\mathbb{B}\Sigma^1_1$ is strictly larger than Σ^1_1 and Π^1_1 since it contains the problems which are in Π^1_1, but not in Σ^1_1, and also the ones which are in Σ^1_1, but not in Π^1_1. We will show that buffered simulation is not only hard for the classes Σ^1_1 and Π^1_1, but also for $\mathbb{B}\Sigma^1_1$.

For any decision problem $P_1 \wedge P_2$ where P_1 and P_2 are in Σ^1_1, we can reduce it to the buffered simulation game. First note that by Corollary 4.4.13, we can construct two pairs of automata $\mathcal{A}_1$, $\mathcal{B}_1$ and $\mathcal{A}_2$, $\mathcal{B}_2$ such that for all $i \in \{1, 2\}$, Duplicator wins $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$ iff P_i has a positive outcome. Thus we can construct two NBA $\mathcal{A}$ and $\mathcal{B}$ from $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}_1, \mathcal{B}_2$ respectively, such that in the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$, Spoiler can force the play to proceed to the component that corresponds to $\mathcal{A}_1, \mathcal{B}_1$ or $\mathcal{A}_2, \mathcal{B}_2$. If both P_1 and P_2 have positive outcomes then no matter how Spoiler plays, Duplicator can win the play. On the other hand, if one of P_1 and P_2 has a negative outcome, suppose P_1, then Spoiler can force the play to proceed to $\mathcal{A}_1$, $\mathcal{B}_1$ and win the play.

Lemma 4.4.14. *Let P_1, $P_2 \in \Sigma^1_1$. There are two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ such that $P_1 \wedge P_2$ has a positive outcome iff $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$.*

Proof. Since P_1, $P_2 \in \Sigma^1_1$, by Corollary 4.4.13, we can reduce them to the problem of deciding $\sqsubseteq^{\omega,0}$. There are two pairs of NBA $\mathcal{A}_1$, $\mathcal{B}_1$ and $\mathcal{A}_2$, $\mathcal{B}_2$ such that for all $i \in \{1, 2\}$, $\mathcal{A}_i \sqsubseteq^{\omega,0} \mathcal{B}_i$ iff P_i has a positive outcome. Supose $\mathcal{A}_i, \mathcal{B}_i$ are defined over a distributed alphabet $\hat{\Sigma}_i = (\Sigma_{i,1}, \Sigma_{i,2})$. Consider two NBA $\mathcal{A}$ and $\mathcal{B}$ that are defined over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ where

$$\Sigma_1 = \Sigma_{1,1} \cup \Sigma_{2,1},$$

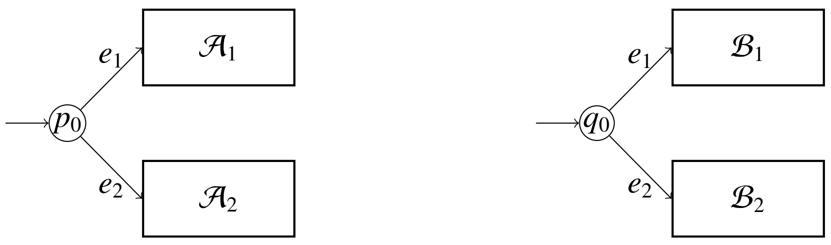


Figure 4.15: Reduction from $P_1 \wedge P_2$ to $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$.

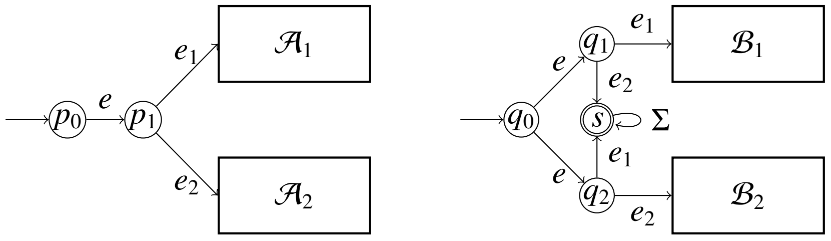


Figure 4.16: Reduction from $P_1 \vee P_2$ to $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$.

$$\Sigma_2 = \Sigma_{1,2} \cup \Sigma_{2,2} \cup \{e_1, e_2\}),$$

and e_1, e_2 are two additional new letters that are not in $\Sigma_{i,j}$ for all $i, j \in \{1,2\}$. The automata $\mathcal{A}$, $\mathcal{B}$ are constructed from $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}_1, \mathcal{B}_2$, respectively, as we have illustrated in Figure 4.15.

Now consider the game $\mathcal{G}^{\omega,0}(\mathcal{A},\mathcal{B})$. If $P_1 \wedge P_2$ has a positive outcome then in the first round, after Spoiler proceeds to some configuration $(p_0^i, (\epsilon, e_i), q_0, \bot, \mathsf{D})$ where p_0^i is the initial state of $\mathcal{A}_i$ and $i \in \{1,2\}$, Duplicator has no choice except to pop e_i from the second buffer and go to q_0^i, the initial state of $\mathcal{B}_i$. Hence the play proceeds to $(p_0^i, (\epsilon, \epsilon), q_0^i, \bot, \mathsf{S})$, which are the initial configuration of the game $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$. Since P_i has a positive outcome for all $i \in \{1,2\}$, from such a configuration, Duplicator can play according to the winning strategy in $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$ and win the play. On the other hand, if $P_1 \wedge P_2$ has a negative outcome, let $i_0 \in \{1,2\}$ such that P_{i_0} has a negative outcome. In the first round, Spoiler reads e_{i_0}, and hence proceeds to $(p_0^{i_0}, (\epsilon, e_{i_0}), q_0, \bot, \mathsf{D})$. Duplicator then will continue to $(p_0^{i_0}, (\epsilon, \epsilon), q_0^{i_0}, \bot, \mathsf{S})$. Since this corresponds to the initial configuration of the game $\mathcal{G}^{\omega,0}(\mathcal{A}_{i_0}, \mathcal{B}_{i_0})$, Spoiler can play according to the winning strategy in $\mathcal{G}^{\omega,0}(\mathcal{A}_{i_0}, \mathcal{B}_{i_0})$ and win the play. □

We can also show similarly that any decision problem $P_1 \vee P_2$ where P_1 and P_2 are in Σ_1^1, can be reduced to buffered simulation game. Consider again the two pairs of automata $\mathcal{A}_1, \mathcal{B}_1$ and $\mathcal{A}_2, \mathcal{B}_2$ where for all $i \in \{1,2\}$, Duplicator wins $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$ iff P_i has a positive outcome. We then construct two NBA $\mathcal{A}$ and $\mathcal{B}$ from $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}_1, \mathcal{B}_2$ respectively, such that in the game $\mathcal{G}^{\omega,0}(\mathcal{A},\mathcal{B})$, Duplicator is the player that chooses whether to proceed to $\mathcal{A}_1, \mathcal{B}_1$ or $\mathcal{A}_2, \mathcal{B}_2$. This is possible by using a similar structure as the F-structure in the initial part of $\mathcal{A}$, $\mathcal{B}$. If one of P_1 and P_2 has a positive outcome, suppose P_1, then Duplicator proceeds to $\mathcal{A}_1$, $\mathcal{B}_1$ and wins the play. On the other hand, if both P_1 and P_2 have negative outcomes then no matter how Duplicator plays, Spoiler wins.

Lemma 4.4.15. *Let P_1, $P_2 \in \Sigma_1^1$. There are two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ such that $P_1 \vee P_2$ has a positive outcome iff $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$.*

Proof. Again, since P_1, $P_2 \in \Sigma_1^1$, by Corollary 4.4.13, we can reduce them to the problem of deciding buffered simulation $\sqsubseteq^{\omega,0}$. There are two pairs of NBA $\mathcal{A}_1$, $\mathcal{B}_1$ and $\mathcal{A}_2$, $\mathcal{B}_2$

such that for all $i \in \{1,2\}$, $\mathcal{A}_i \sqsubseteq^{\omega,0} \mathcal{B}_i$ iff P_i has a positive outcome. Suppose $\mathcal{A}_i, \mathcal{B}_i$ are over the distributed alphabet $\hat{\Sigma}_i = (\Sigma_{i,1}, \Sigma_{i,2})$. Consider two NBA $\mathcal{A}$ and $\mathcal{B}$ that are defined over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ where

$$\begin{aligned}\Sigma_1 &= \Sigma_{1,1} \cup \Sigma_{2,1},\\ \Sigma_2 &= \Sigma_{1,2} \cup \Sigma_{2,2} \cup \{e, e_1, e_2\}),\end{aligned}$$

and e, e_1, e_2 are three additional new letters that are not in $\Sigma_{i,j}$ for all $i, j \in \{1,2\}$. The automata $\mathcal{A}$, $\mathcal{B}$ are constructed from $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{B}_1, \mathcal{B}_2$ as illustrated in Figure 4.16.

Now consider the game $\mathcal{G}^{\omega,0}(\mathcal{A}, \mathcal{B})$. If $P_1 \vee P_2$ has a positive outcome then in the first round, after Spoiler proceeds to $(p_1, (\epsilon, e), q_0, \perp, \mathsf{D})$, Duplicator goes to q_i where P_i has a positive outcome. Hence the play proceeds to the configuration $(p_1, (\epsilon, \epsilon), q_i, \perp, \mathsf{S})$. If Spoiler reads $e_{\bar{i}}$ where $\bar{i} \in \{1,2\} \setminus \{i\}$, Duplicator goes to the accepting sink s by reading $e_{\bar{i}}$. Otherwise, Spoiler reads e_i and in this case, Duplicator proceeds to the configuration $(p_0^i, (\epsilon, \epsilon), q_0^i, \perp, \mathsf{S})$ where p_0^i, q_0^i are the initial states of $\mathcal{A}_i$, $\mathcal{B}_i$. Such a configuration corresponds to the initial configuration of the game $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$. Since P_i has a positive outcome, from such a configuration, Duplicator can play according to the winning strategy in $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$ and win the play. Now suppose $P_1 \vee P_2$ has a negative outcome. The strategy for Spoiler is as follows. If in the first round, after Spoiler reads e, Duplicator goes to some state q_i, $i \in \{1,2\}$, Spoiler then continues the play by reading e_i. The play then proceeds to some configuration $(p_0^i, (\epsilon, \epsilon), q_0^i, \perp, \mathsf{S})$ where p_0^i, q_0^i are the initial states of $\mathcal{A}_i$, $\mathcal{B}_i$. Such a configuration corresponds to the initial configuration of the game $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$. Since for all $i \in \{1,2\}$, P_i has a negative outcome, Spoiler can play according to the winning strategy in $\mathcal{G}^{\omega,0}(\mathcal{A}_i, \mathcal{B}_i)$ and win the play. □

By considering a similar reduction as in Lemma 4.4.14 and Lemma 4.4.15, we can show that any problem in $\mathbb{B}\Sigma_1^1$ can be reduced to a buffered simulation game.

Theorem 4.4.16. *For any decision problem $P \in \mathbb{B}\Sigma_1^1$, there are two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$ such that P has a positive outcome iff $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$.*

Proof. Let $P \in \mathbb{B}\Sigma_1^1$. Without loss of generality, let us assume that P is in normal form where negation is only applied to the atomic problems. We will show that the property holds by induction on the structure of P. If $P \in \Sigma_1^1$ then by Corollary 4.4.13 we have the desired result. If P is $\neg P'$ and $P' \in \Sigma_1^1$ then $P \in \Pi_1^1$. By Corollary 4.4.6 we also have the desired result. If P is $P_1 \wedge P_2$ and $P_1, P_2 \in \mathbb{B}\Sigma_1^1$, by induction hypothesis, there are two pairs of NBA $\mathcal{A}_1, \mathcal{B}_1$ and $\mathcal{A}_2, \mathcal{B}_2$ such that for all $i \in \{1,2\}$, the problem P_i has a positive outcome iff $\mathcal{A}_i \sqsubseteq^{\omega,0} \mathcal{B}$. We can construct two NBA $\mathcal{A}$, $\mathcal{B}$ as in the proof of Lemma 4.4.14 such that $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$ iff P has a positive outcome. Similarly, if P is $P_1 \vee P_2$ and $P_1, P_2 \in \mathbb{B}\Sigma_1^1$, by induction hypothesis, there are two pairs of NBA $\mathcal{A}_1, \mathcal{B}_1$ and $\mathcal{A}_2, \mathcal{B}_2$ such that for all $i \in \{1,2\}$, the problem P_i has a positive outcome iff $\mathcal{A}_i \sqsubseteq^{\omega,0} \mathcal{B}_i$. We can construct two NBA $\mathcal{A}$, $\mathcal{B}$ as in the proof of Lemma 4.4.15 such that $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$ iff P has a positive outcome. □

Since every problems in $\mathbb{B}\Sigma_1^1$ can be reduced to the problem of deciding buffered simulation $\sqsubseteq^{\omega,0}$, we have the following theorem.

Theorem 4.4.17. *Deciding $\sqsubseteq^{\omega,0}$ is $\mathbb{B}\Sigma_1^1$-hard.*

In the following subsection, we will show that the problem of deciding whether Duplicator wins the buffered simulation game $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ for two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma}$ =

$(\Sigma_1, \dots, \Sigma_n)$ and a $\kappa = (k_1, \dots, k_n)$ is indeed in the second level of the analytical hierarchy. Such a problem is in the class $\Sigma_2^1 \cap \Pi_2^1$, a class that does not contain any complete problem [RJ87].

4.4.4 Membership in $\Sigma_2^1 \cap \Pi_2^1$

To complete the undecidability result of buffered simulation, we will show that for any two NBA over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ and a capacity vector $\kappa = (k_1, \dots, k_n)$, the problem of deciding $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ is in the classes Σ_2^1 and Π_2^1. Recall that to show such memberships, we have to show that the problem can be characterised by second-order formulae of the form $\forall X \, \exists Y \, \phi_1(X, Y)$ and $\exists X \, \forall Y \, \phi_2(X, Y)$ where ϕ_1 and ϕ_2 are first-order formulae [RJ87, Kec95].

Before we show the formulae, we will give the predicates that are used to construct them. First, consider the game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B}) = ((V, V_\mathsf{D}, V_\mathsf{S}, E), v_0, \mathsf{Win})$. A valid finite play in such a game is nonetheless a path $r \in V^+$ in the configuration graph of $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$. Hence the following is the predicate that defines the valid finite plays in $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$.

$$\mathsf{FinPlay}(r) := \forall i \in \{1, \dots, |r|\} \; V(r_i) \wedge E(r_i, r_{i+1}) \wedge v_0(r_1) \tag{4.30}$$

Furthermore, a valid finite play r is played according to some Spoiler's or Duplicator's strategy σ_S or σ_D, respectively, if for all $i \in \{1, \dots, n\}$, whenever $r_i \in V_x$, $x \in \{\mathsf{S}, \mathsf{D}\}$, then r_{i+1} is $\sigma_x(r_1 \dots r_i)$. Hence the following are the predicates that define valid finite plays that are played according to Spoiler's strategy σ_S and Duplicator's strategy σ_D, respectively.

$$\mathsf{FinConsistent}_\mathsf{S}(r, \sigma_\mathsf{S}) := \forall i \in \{1, \dots, |r|\} \; V_\mathsf{S}(r_i) \Rightarrow r_{i+1} = \sigma_\mathsf{S}(r_1 \dots r_i) \tag{4.31}$$

$$\mathsf{FinConsistent}_\mathsf{D}(r, \sigma_\mathsf{D}) := \forall i \in \{1, \dots, |r|\} \; V_\mathsf{D}(r_i) \Rightarrow r_{i+1} = \sigma_\mathsf{D}(r_1 \dots r_i) \tag{4.32}$$

We can also lift the decidable predicates in (4.30) - (4.32) for the infinite case. The following are the predicates that respectively define valid infinite plays, infinite plays that are played according to some Spoiler's strategy σ_S, and the ones that are played according to some Duplicator's strategy σ_D.

$$\begin{aligned} \mathsf{InfPlay}(\pi) &:= \forall i \in \mathbb{N} \; V(\pi_i) \wedge E(\pi_i, \pi_{i+1}) \wedge v_0(\pi_0) \\ \mathsf{Consistent}_\mathsf{S}(\pi, \sigma_\mathsf{S}) &:= \forall i \in \mathbb{N} \; V_\mathsf{S}(\pi_i) \Rightarrow \pi_{i+1} = \sigma_\mathsf{S}(\pi_0 \dots \pi_i) \\ \mathsf{Consistent}_\mathsf{D}(\pi, \sigma_\mathsf{D}) &:= \forall i \in \mathbb{N} \; V_\mathsf{D}(\pi_i) \Rightarrow \pi_{i+1} = \sigma_\mathsf{D}(\pi_0 \dots \pi_i) \end{aligned}$$

These predicates are of the form $\forall x \, \psi(x)$ where $\psi(x)$ is quantifier-free. Hence the problem of deciding whether an infinite play is valid or whether a play is played according to some Spoiler's or Duplicator's strategy belongs to the class Π_1^0.

Now let F_D be the set of Spoiler's configurations in which the fourth component is $\top$, and F_S the set of Duplicator's configurations in which the first component is an accepting state, i.e. $F_\mathsf{D} = \{(p, \overline{w}, q, c, \mathsf{S}) \in V_S \mid c = \top\}$ and $F_\mathsf{S} = \{(p, \overline{w}, q, c, \mathsf{D}) \in V_D \mid p \in F^\mathcal{A}\}$. Intuitively, F_D and F_S are the sets of configurations which are obtained from Spoiler's and Duplicator's moves through some accepting state. An infinite play $\pi = v_0 v_1 \dots$ then is winning for Duplicator if either there are infinitely many i such that $v_i \in F_\mathsf{D}$ or there are only finitely many i such that $v_i \in F_\mathsf{S}$. Hence the following predicate defines plays that are winning for Duplicator.

$$\mathsf{Win}_\mathsf{D}(\pi) := \exists i \in \mathbb{N} \;\; \forall j > i \;\; \exists k > j \;\; \neg F_\mathsf{S}(\pi_j) \wedge F_\mathsf{D}(\pi_k)$$

Intuitively, it says that Duplicator wins a play π if at one point, we do not see configurations from F_{S} any more, but we keep seeing the ones from F_{D}. The predicate is of the form $\exists x \forall y \exists z\, \psi(x)$ where $\psi(x)$ is quantifier-free. Hence the problem of deciding whether an infinite play is winning for Duplicator belongs to the class Σ^0_3.

Deciding $\sqsubseteq^\kappa$ is in Σ^1_2

Now we will give a formula that defines buffered simulation $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ by using all predicates that we have given before. Recall that Duplicator wins $\mathcal{G}^\kappa(\mathcal{A},\mathcal{B})$ iff there exists Duplicator's strategy such that for every valid play that is played according to this strategy, either it is an infinite play and winning for Duplicator, or it is a finite play where Spoiler eventually gets stuck. Hence we can consider the following formula.

$$\begin{aligned}\sqsubseteq^\kappa(\mathcal{A},\mathcal{B}) := \exists \sigma_{\mathsf{D}}\, \forall \pi,\ & \\ & (\mathsf{InfPlay}(\pi) \land \mathsf{Consistent}_{\mathsf{D}}(\pi,\sigma_{\mathsf{D}}) \Rightarrow \mathsf{Win}_{\mathsf{D}}(\pi)) \\ & \lor (\exists i\ \neg\mathsf{FinPlay}(\pi_0 \ldots \pi_{i+1}) \land \mathsf{FinPlay}(\pi_0 \ldots \pi_i) \\ & \qquad \land \mathsf{FinConsistent}_{\mathsf{D}}(\pi_0 \ldots \pi_i, \sigma_{\mathsf{D}}) \Rightarrow V_{\mathsf{S}}(\pi_i))\end{aligned}$$

Note that this predicate is of the form $\exists X \forall Y\ \psi(X,Y)$ where $\psi(X,Y)$ is a first order formula and all of its atomic predicates are decidable. Hence we have the following lemma.

Lemma 4.4.18. *Deciding $\sqsubseteq^\kappa$ is in Σ^1_2.*

Deciding $\sqsubseteq^\kappa$ is in Π^1_2

We can also define buffered simulation with another formula. It is also the case that Duplicator wins $\mathcal{G}^\kappa(\mathcal{A},\mathcal{B})$ iff for every Spoiler's strategy, there exists a valid play that is played according to the strategy such that either the play is infinite and winning for Duplicator or it is a finite play where Spoiler eventually gets stuck. Hence we can also define buffered simulation with the following formula.

$$\begin{aligned}\sqsubseteq^\kappa(\mathcal{A},\mathcal{B}) := \forall \sigma_{\mathsf{S}}\, \exists \pi\ & \\ & (\mathsf{InfPlay}(\pi) \land \mathsf{Consistent}_{\mathsf{S}}(\pi,\sigma_{\mathsf{S}}) \Rightarrow \mathsf{Win}_{\mathsf{D}}(\pi)) \\ & \lor (\exists i\ \neg\mathsf{FinPlay}(\pi_0 \ldots \pi_{i+1}) \land \mathsf{FinPlay}(\pi_0 \ldots \pi_i) \\ & \qquad \land \mathsf{FinConsistent}_{\mathsf{S}}(\pi_0 \ldots \pi_i, \sigma_{\mathsf{S}}) \Rightarrow V_{\mathsf{S}}(\pi_i))\end{aligned}$$

The formula is of the form $\forall X \exists X\ \psi(X,Y)$ where $\psi(X,Y)$ is a first order formula and its atomic predicates are also decidable. Hence the problem of deciding whether Duplicator wins $\mathcal{G}^\kappa(\mathcal{A},\mathcal{B})$ is also in Π^1_2.

We can put this together with Lemma 4.4.18 and have the following theorem.

Theorem 4.4.19. *Deciding $\sqsubseteq^\kappa$ is in Δ^1_2.*

4.5 Summary

We summarise the complexity of buffered simulation that is shown in this chapter in Figure 4.17. In the case where we only consider one buffer with a fixed and bounded capacity $k \in \mathbb{N}$, buffered simulation and its flushing variant can be solved in polynomial time. This result can be lifted to the case of multiple buffers. Buffered simulation with

Problem		Complexity
Given	Question	
$\mathcal{A}$, $\mathcal{B}$ over Σ and a fixed $k \in \mathbb{N}$	Is $\mathcal{A} \sqsubseteq^{k} \mathcal{B}$? Is $\mathcal{A} \sqsubseteq^{k}_{\mathsf{Flush}} \mathcal{B}$?	PTIME Cor. 4.1.6, Cor. 4.2.5
$\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a fixed $\kappa \in \mathbb{N}^n$	Is $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$? Is $\mathcal{A} \sqsubseteq^{\kappa}_{\mathsf{Flush}} \mathcal{B}$?	
$\mathcal{A}$, $\mathcal{B}$ over Σ	Is $\mathcal{A} \sqsubseteq^{\omega}_{\mathsf{Flush}} \mathcal{B}$?	PSPACE-complete Cor. 4.3.17
	Is $\mathcal{A} \sqsubseteq^{\omega} \mathcal{B}$?	EXPTIME-complete Cor. 4.3.25
$\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \Sigma_2)$	Is $\mathcal{A} \sqsubseteq^{\omega,0} \mathcal{B}$?	in Δ^1_2 and $\mathbb{B}\Sigma^1_1$-hard Thm. 4.4.17, Thm. 4.4.19
$\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a fixed $\kappa \in (\mathbb{N} \cup \{\omega\})^n$	Is $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$?	

Figure 4.17: Complexity of deciding buffered simulation.

$n \geq 1$ buffers and a fixed capacity vector $\kappa = (k_1, \ldots, k_n) \in \mathbb{N}^+$ can be solved in polynomial time. In the case where we consider unbounded buffers, solving buffered simulation where only one buffer is involved is PSPACE-complete for the flushing variant and EXPTIME-complete for the general case. However if multiple buffers are involved then buffered simulation is undecidable. It is in the class of Δ^1_2 and hard for the class $\mathbb{B}\Sigma^1_1$. This high undecidability is true already for the case where we only have two buffers in which one is unbounded and the other one is of capacity 0.

Chapter 5

Application to Formal Languages

In this chapter, we will present the application of buffered simulation in the field of formal languages. In the case where we only have one buffer, buffered simulation approximates language inclusion between two Büchi automata in the same way as the standard fair simulation. If Duplicator wins the buffered simulation game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N} \cup \{\omega\}$ then we have language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Intuitively, the possibility to use a buffer gives Duplicator more power to mimic Spoiler's run. The bigger the buffer the more power Duplicator has to show language inclusion. Hence buffered simulation gets closer to language inclusion as the size of the buffer grows.

In the case where we consider multiple buffers, buffered simulation can be used to approximate a more general problem than language inclusion, namely Mazurkiewicz trace inclusion. Mazurkiewicz traces, or just traces, basically extend the concept of words, in which some letters are allowed to commute and some are not [Maz77, DR95]. They are used to model the computation of concurrent systems. The problem of deciding the trace closure inclusion of ω-languages recognised by Büchi automata is known to be highly undecidable [Sak92, Fin12]. However, we can use buffered simulation to approximate this problem. For any two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$, if Duplicator wins the buffered simulation game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$ for some $\kappa \in (\mathbb{N} \cup \{\omega\})^n$ then we have trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$.

The approximation of language or trace closure inclusion with buffered simulation however is not complete. There are pairs of automata where language or trace closure inclusion holds, but buffered simulation does not. Hence one may ask whether there is a characteristic of pairs of automata in which their language or trace closure inclusion cannot be shown by buffered simulation. The answer to this question turns out to be related to the notion of continuity from the field of topology. First note that language or trace closure inclusion can be characterised by the existence of a function that maps each accepting run in Spoiler's automaton to a corresponding accepting run in Duplicator's automaton. It turns out that in the case where we have such a function that is also continuous, language or trace closure inclusion also implies buffered simulation. The reverse direction of this property indeed also holds. Language or trace closure inclusion can be shown with buffered simulation iff such a continuous function exists. Intuitively, we should be able to lift this characterisation to the case of bounded buffers by considering a function that is not only continuous, but also Lipschitz continuous. However we will show that the characterisation with a Lipschitz continuous function does not hold in general, but only for some more restricted automata.

This chapter is organised as follows. The first section shows the use of buffered sim-

Algorithm 2 Checking $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

```
1: $k \leftarrow 0$
2: while $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$ do
3:     $k \leftarrow k + 1$
4: end while
5: return yes
```

ulation with one buffer to approximate language inclusion incrementally. The second one considers the application of buffered simulation with multiple buffers to approximate trace closure inclusion. We first recall the theory of Mazurkiewicz traces, the trace closure inclusion problem, and its high undecidability. We then show that buffered simulation with multiple buffers can be used to approximate such a problem. The last part of this chapter considers the characterisation of buffered simulation with the notion of continuity. We first show the characterisation of buffered simulation with a continuous function and then the refined characterisation of buffered simulation with bounded buffers with a Lipschitz continuous function which only holds for *cyclic-path-connected* automata.

5.1 The Language Inclusion Problem

First let us consider the application of buffered simulation with one buffer for the language inclusion problem. Language inclusion between two ω-regular languages represented by two NBA $\mathcal{A}$, $\mathcal{B}$ is known to be an important problem in the area of formal languages [VW86, Var96]. Such a problem can be used to model the verification problem of non-terminating reactive system. The system that we want to verify is modeled by an automaton $\mathcal{A}$ and the specification that has to be met by the system is modeled by an automaton $\mathcal{B}$. The problem of checking whether the system meets the specification then is reduced to the problem of checking language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. This problem is unfortunately hard to compute, i.e. it is PSPACE-complete.

5.1.1 Incremental Approximation

Buffered simulation with one buffer can be used to approximate language inclusion in the same sense as the standard fair simulation.

Theorem 5.1.1. *If there exists $k \in \mathbb{N} \cup \{\omega\}$ such that $\mathcal{A} \sqsubseteq^k \mathcal{B}$ then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.*

Proof. Suppose $w \in L(\mathcal{A})$. Since w is accepted by $\mathcal{A}$, there exists an accepting run $\rho \in \mathsf{AccRun}(\mathcal{A})$ over w. Let $k \in \mathbb{N} \cup \{\omega\}$ such that $\mathcal{A} \sqsubseteq^k \mathcal{B}$. Since Duplicator wins $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$, if Spoiler plays ρ and Duplicator plays according to the winning strategy then Duplicator forms an accepting run $\rho' \in \mathsf{AccRun}(\mathcal{B})$. Suppose $w = a_1 a_2 \ldots$. Hence $a_1, a_2, \ldots$ are the letters that are pushed to the buffer consecutively. Since Duplicator pops every letter in this order, we have $\mathsf{word}(\rho') = a_1 a_2 \ldots$. Thus $w \in L(\mathcal{B})$. □

We can show language inclusion by using a buffered simulation game with one buffer. Note that Duplicator gets stronger in showing language inclusion as the buffer capacity grows. Recall from Theorem 3.1.4 and Proposition 3.1.6 that buffered simulation with one buffer admits a hierarchy, i.e. we have

$$\sqsubseteq^0 \subsetneq \sqsubseteq^1 \subsetneq \ldots \subsetneq \sqsubseteq^\omega.$$

Hence as the capacity grows, there are more pairs of automata for which their language inclusion can be shown by buffered simulation. This then allows us to use buffered simulation as an incremental tool for approximating language inclusion. To show language inclusion between two automata $\mathcal{A}$ and $\mathcal{B}$, we start from some small capacity $k \in \mathbb{N}$ and then check whether $\mathcal{A} \sqsubseteq^k \mathcal{B}$ holds. If yes then we conclude that language inclusion holds. Otherwise we increase k and then check again. We illustrate such a procedure in Algorithm 2.

Drawback of Algorithm 2 One of the biggest drawbacks of Algorithm 2 is that the algorithm may not terminate. For example, it does not terminate on two NBA $\mathcal{A}$, $\mathcal{B}$ where $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$. For such $\mathcal{A}$, $\mathcal{B}$, by Theorem 5.1.1, we know that $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$ for any $k \in \mathbb{N}$. Hence for any two NBA where language inclusion does not hold, the algorithm runs forever. The algorithm might also not terminate on $\mathcal{A}$, $\mathcal{B}$ in which $L(\mathcal{A}) \subseteq L(\mathcal{B})$. For example, consider the two NBA $\mathcal{A}$, $\mathcal{B}$ from Example 3.1.5. In this case, we have $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but we have seen that $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$ for any $k \in \mathbb{N}$. For such inputs, Algorithm 2 also does not terminate.

Complexity of Algorithm 2 In Theorem 4.1.2 we have seen that deciding buffered simulation $\mathcal{A} \sqsubseteq^k \mathcal{B}$ is in time $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{3k} \cdot k^3)$. Hence each while-loop iteration in Algorithm 2 runs in polynomial time. However note that for each while-loop iteration, the size of the buffer, i.e. k, is incremented by one. Hence as k grows, the complexity of solving buffered simulation $\mathcal{A} \sqsubseteq^k \mathcal{B}$ also grows exponentially in k.

5.1.2 Comparison

The flushing variant of buffered simulation can also be used to approximate language inclusion incrementally. We can consider a similar procedure as in Algorithm 2 where in each while-loop iteration, instead of checking $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$, we check $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$ or $\mathcal{A} \not\sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$.

The incremental algorithms induced by the flushing and the full-flushing variants also have the same drawbacks as Algorithm 2. They do not terminate on $\mathcal{A}$, $\mathcal{B}$ where $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$, and on $\mathcal{A}$, $\mathcal{B}$ in which $L(\mathcal{A}) \subseteq L(\mathcal{B})$, but for all $k \in \mathbb{N}$, $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$ or $\mathcal{A} \not\sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$.

The incremental algorithms induced by the flushing and the full-flushing variants however run slightly better than the general case. Recall that for a fixed $k \in \mathbb{N}$, deciding $\mathcal{A} \sqsubseteq^k_{\mathsf{Flush}} \mathcal{B}$ and $\mathcal{A} \sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$ are respectively $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{2k+4})$ and $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^3 \cdot |\Sigma|^{2k+2})$. Hence the base of the exponent in these cases is $|\Sigma|^2$, while in the general case it is $|\Sigma|^3$. Since the base of the exponent is smaller than in the general case, as k grows, the complexity of the incremental algorithm induced by the flushing or the full-flushing variant grows slower than the general case.

There is however one disadvantage of the incremental algorithm induced by the full-flushing variant in comparison to the one induced by the flushing variant or the general case. Recall from Chapter 3 that the full-flushing variant does not admit a hierarchy. We have seen in Theorem 3.2.6 that for any $k \in \mathbb{N}$, there exist $\mathcal{A}$, $\mathcal{B}$ such that $\mathcal{A} \sqsubseteq^k_{\mathsf{FFlush}} \mathcal{B}$, but $\mathcal{A} \not\sqsubseteq^{k+1}_{\mathsf{FFlush}} \mathcal{B}$. Hence Duplicator does not get stronger in winning $\mathcal{G}^k_{\mathsf{FFlush}}(\mathcal{A}, \mathcal{B})$ as the capacity k grows. In this case, determining whether increasing the parameter k will not help us anymore is harder than in the case of the flushing variant or the general case of buffered simulation.

Algorithm 3 Checking $L(\mathcal{A}) \subseteq L(\mathcal{B})$ with Multi-Pebble Simulation.

```
1: $k \leftarrow 0, n \leftarrow |\mathcal{B}|$
2: while $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ and $k \leq n$ do
3:     $k \leftarrow k + 1$
4: end while
5: If $k \leq n$ return yes otherwise don't know
```

Comparison with the Multi-Pebble Incremental Approach

Another possible incremental approach for language inclusion is multi-pebble simulation. As we have seen in Chapter 2, multi-pebble simulation also approximates language inclusion in the same way as standard fair simulation. Recall that by Proposition 2.6.20, if Duplicator wins the game $\mathcal{G}^k_{\mathsf{Peb}}(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N}$ then we have language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Moreover in [Ete02], it is also shown that multi-pebble simulation also admits a hierarchy in the same sense as buffered simulation. For any $k \geq 1$, if we have $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ then this implies $\mathcal{A} \sqsubseteq^{k+1}_{\mathsf{Peb}} \mathcal{B}$.

Proposition 5.1.2 ([Ete02])**.** $\sqsubseteq^1_{\mathsf{Peb}} \subseteq \sqsubseteq^2_{\mathsf{Peb}} \subseteq \sqsubseteq^3_{\mathsf{Peb}} \subseteq \ \ldots$

This proposition shows that as the number of the pebbles grows, Duplicator gets stronger in winning the multi-pebble simulation game. Hence multi-pebble simulation also gets closer to the language inclusion as the number of pebbles grows. We can use multi-pebble simulation to incrementally approximate language inclusion as in the case of buffered simulation.

There is, however, one advantage of using multi-pebble simulation. Note that when the number of pebbles is equal to the number of states in Duplicator's automaton then adding more pebbles does not give any advantage for Duplicator to win the game. We have the following proposition.

Proposition 5.1.3 ([Ete02])**.** *If $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ for some $k > 0$ then $\mathcal{A} \sqsubseteq^{|\mathcal{B}|}_{\mathsf{Peb}} \mathcal{B}$.*

This allows us to stop increasing the parameter k if it reaches the size of Duplicator's automaton. Hence unlike buffered simulation and its flushing variant, multi-pebble simulation induces an incremental approach that always terminates.

Consider the incremental algorithm given in Algorithm 3. Given two NBA $\mathcal{A}$, $\mathcal{B}$, we start with $k = 0$ and check whether $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$. If this is not the case then we gradually increase the parameter k by one. If we reach $k = |\mathcal{B}|$ and we still have $\mathcal{A} \not\sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ then we stop and conclude that we do not know whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds. However, if we eventually have $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ for some $k \leq n$ then we conclude that language inclusion holds.

Complexity of Algorithm 3 In comparison to Algorithm 2, Algorithm 3 has a drawback with respect to its complexity. First note that by Proposition 2.6.23, for any two NBA $\mathcal{A}$, $\mathcal{B}$, and a $k \in \mathbb{N}$, deciding $\mathcal{A} \sqsubseteq^k_{\mathsf{Peb}} \mathcal{B}$ can be done in time $O(|\mathcal{A}|^3 \cdot |\mathcal{B}|^{3k} \cdot |\Sigma|^3)$. Hence as k grows, the complexity of solving multi-pebble simulation also grows exponentially in k. However note that the base of the exponent is a variable. It is the size of the automaton $\mathcal{B}$ which is part of the input and usually very large. This is in contrast to the incremental algorithm induced by buffered simulation. As k grows, the complexity of solving buffered simulation also grows exponentially in k but the base of the exponent is fixed. It is the size of the alphabet Σ which is usually small and not part of the input. Hence the complexity of solving buffered simulation in each while-loop iteration of Algorithm 2 is expected to grow much slower than those of multi-pebble simulation.

5.2 The Trace Closure Inclusion Problem

Another possible application of buffered simulation comes from the area of Mazurkiewicz traces [DR95]. Mazurkiewicz traces, or just traces, are used to model concurrent computations. In a concurrent computation, two processes can be dependent on or independent of each other. If two processes are dependent on each other then the order in which one should be executed first is important; one process might need the output of the other one. On the other hand, if two processes are independent of each other then the order in which they should be executed does not matter. They can be executed simultaneously and the output is still the same.

The theory of Mazurkiewicz traces is used to provide a mathematical model for concurent computations. A process is modeled by a letter and the independency between the processes is modeled by the independency between the letters. A concurrent computation then can be linearly modeled by a word where two adjacent independent letters are allowed to commute with each other. For example, suppose we consider words over a, b, and c where a, b are independent of each other, but not of c: the letters a, b can commute with each other, but they cannot commute with c. The finite word abc then is considered to be equivalent to bac. Both of abc and bac correspond to a concurrent computation of a and b followed by the execution of c. Note that this concept can also be naturally extended to infinite words. For example, if we consider the same independency between a, b, c then the infinite word $abab\ldots$ is considered to be equal to the infinite word $baba\ldots$ Both of these words model the concurrent computation of $aaa\ldots$ and $bbb\ldots$. We will recall the formal definition of Mazurkiewicz traces and trace closure inclusion in the first part of this section and the application of buffered simulation for Mazurkiewicz traces in the second one.

5.2.1 Mazurkiewicz Traces

Formally, traces are defined over a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. Given a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ be the union of the alphabets and π_i, $i \in \{1, \ldots, n\}$, be the projection with respect to $\hat{\Sigma}$ as in Definition 3.4.1. Two words $v, w \in \Sigma^\infty$ are said to be *trace equivalent*, written $v \sim w$, iff $\pi_i(v) = \pi_i(w)$ for all $i \in \{1, \ldots, n\}$. Note that in the finite case, two words are trace equivalent only if they are of the same length. If they are not then there must be a letter a such that $|v|_a \neq |w|_a$ which implies $\pi_i(v) \neq \pi_i(w)$ for some $i \in \{1, \ldots, n\}$ where $a \in \Sigma_i$. We exemplify trace equivalence in the following example.

Example 5.2.1. Consider a distributed alphabet $\hat{\Sigma} = (\{a, c\}, \{b, c\})$. We have $abc \nsim abcc$ since $\pi_1(abc) = ac \neq acc = \pi_1(abcc)$. However we have $abc \sim bac$ since $\pi_1(abc) = \pi_1(bac) = ac$ and $\pi_2(abc) = \pi_2(bac) = bc$. We also have $(ab)^\omega \sim (ba)^\omega$ since $\pi_1((ab)^\omega) = \pi_1((ba)^\omega) = a^\omega$ and $\pi_2((ab)^\omega) = \pi_2((ba)^\omega) = b^\omega$. Note that for any word $w \in (a \cup b)^\omega$ in which $|w|_a = |w|_b = \infty$, we indeed have $w \sim (ab)^\omega$ since $\pi_1(w) = a^\omega$ and $\pi_2(w) = b^\omega$.

By definition of trace equivalence, intuitively two letters $a, b \in \Sigma$ can commute with each other iff they do not share any Σ_i, i.e. there is no $i \in \{1, \ldots, n\}$ such that $a, b \in \Sigma_i$.

The equivalence classes of Σ^* with respect to $\sim$ are called *finite traces* and the equivalence classes of Σ^ω with respect to $\sim$ are called *infinite traces*. For any language $L \subseteq \Sigma^\infty$ we denote the trace closure of L by $[L]$. It is the set of words that are trace equivalent to some words in L, i.e.

$$[L] = \{v \in \Sigma^\infty \mid v \sim w,\ w \in L\}.$$

Example 5.2.2. Consider again the distributed alphabet $\hat{\Sigma}$ from Example 5.2.1 and the languages $L_1 = a^*(cb)^*$ and $L_2 = (ab)^\omega$. Since for every $w \in L_1$, there is no w' that is distinct from w such that $w' \sim w$, we have $[L_1] = L_1$. For the language L_2, note that the words that are trace equivalent to $(ab)^\omega$ are the ones over a, b that contain infinitely many a and b. Hence $[L_2] = \{w \in (a \cup b)^\omega \mid |w|_a = |w|_b = \infty\}$.

The definition of trace equivalence that we consider here is defined with respect to a distributed alphabet, a notion due to [Zie87]. The original definition of trace equivalence however is defined with respect to an *independence alphabet*, a pair of alphabet and a relation between the letters that tells us which pair of letters can commute with each other [Maz89].

Formally, an independence alphabet is a pair (Σ, I) where Σ is an alphabet and I is a relation over Σ that is irreflexive and symmetric called *independence relation*. The independence relation defines pairs of letters that commute with each other. For a distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, the corresponding independence alphabet is (Σ, I) where $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ and $I = \{(a, b) \in \Sigma^2 \mid \nexists i \in \{1, \ldots, n\} : a, b \in \Sigma_i\}$.

Example 5.2.3. The corresponding independence alphabet of the distributed alphabet $\hat{\Sigma}$ given in Example 5.2.1 is (Σ, I) where $\Sigma = \{a, b, c\}$ and

$$I = \{(a, b), (b, a)\}.$$

Another possible notion to define trace equivalence is by considering a *dependence alphabet*. A dependence alphabet is a pair (Σ, D) where Σ is an alphabet and D is a relation over Σ that is reflexive and symmetric, called *dependence relation*. In contrast to the independence relation, the dependence relation defines pairs of letters that do not commute with each other. For any independence alphabet (Σ, I), the corresponding dependence alphabet is (Σ, D) where $D = \Sigma^2 \setminus I$.

Example 5.2.4. The corresponding dependence alphabet of (Σ, I) given in Example 5.2.3 is (Σ, D), where

$$D = \{(a, a), (b, b), (c, c), (a, c), (c, a), (c, b), (b, c)\}.$$

Beside the dependence alphabet, one also often uses the notion of *dependency graph*. A dependency graph is an equivalent notion for the dependence alphabet (Σ, D). It is a visualisation of the dependence alphabet where the nodes are letters in Σ and we have an edge between two letters if they are dependent on each other. One important thing in the dependency graph is that self-loops are omitted. For any dependence alphabet (Σ, D), the corresponding dependency graph is the graph $G = (V, E)$ where $V = \Sigma$ and $E = D \setminus \{(q, q) \mid q \in V\}$.

Example 5.2.5. Consider the dependence alphabet (Σ, D) from the previous example. The corresponding dependency graph is the following graph.

$$a - c - b$$

Given a dependency graph G, two letters commute with each other if they are not in the edge relation. For any dependency graph $G = (V, E)$, the corresponding distributed alphabet is the tuple $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ in which $\{\Sigma_1, \ldots, \Sigma_n\}$ is the set of maximal cliques in G. Hence two letters $a, b \in V$ are in the edge relation $(a, b) \in E$ iff there exists $i \in \{1, \ldots, n\}$ such that $a, b \in \Sigma_i$.

Distributed, independence, and dependence alphabets, as well as dependency graphs, are equivalent notions that can be used to define trace equivalence. They can be derived from each other. It is sometimes more convenient to use one of them than the others. For the rest of this work, we will mostly use the definition of trace equivalence with respect to a distributed alphabet.

Now consider the following decision problem.

$$\begin{array}{l}\textbf{Given :} \text{ Two NBA } \mathcal{A}, \mathcal{B} \text{ over } \hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n) \\ \textbf{Question :} \text{ Is } [L(\mathcal{A})] \subseteq [L(\mathcal{B})]?\end{array} \tag{5.1}$$

Such a problem can be used to model the verification problem of reactive system that admits concurrency [Sak92, MSB+16]. The system and the specification that we want to check are modeled respectively by the NBA $\mathcal{A}$ and $\mathcal{B}$ over the distributed alphabet $\hat{\Sigma}$ that models the dependency between the processes of the system. Hence the trace closures $[L(\mathcal{A})]$ and $[L(\mathcal{B})]$ represent all possible concurrent computations that can be executed by the system and the permitted behaviours. The problem of checking whether the concurrent system meets the given specification then can be modeled by the trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$. This problem, however, is known to be highly undecidable.

Proposition 5.2.6 ([Sak92, Fin12]). *Given two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma}$, deciding $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$ is undecidable and it is Π^1_1-hard.*

5.2.2 Incremental Approximation

Buffered simulation with multiple buffers can be used to approximate trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$. Before we show this, first note that the trace closure inclusion $[L_1] \subseteq [L_2]$ is equivalent to the inclusion $L_1 \subseteq [L_2]$.

Lemma 5.2.7. *For any two ω-languages L_1, L_2 over $\hat{\Sigma}$,*

$$[L_1] \subseteq [L_2] \textit{ iff } L_1 \subseteq [L_2].$$

Proof. For the right-to-left direction, suppose $L_1 \subseteq [L_2]$. Let $w \in [L_1]$ be some arbitrary word. There is a word $w' \in L_1$ such that $w' \sim w$. Since $L_1 \subseteq [L_2]$, we have $w' \in [L_2]$. There is a word $w'' \in L_2$ such that $w'' \sim w'$. Since $w \sim w''$ and $w'' \sim w'$, we have $w \in [L_2]$. Hence $[L_1] \subseteq [L_2]$. The left-to-right direction simply holds because $L_1 \subseteq [L_1]$. Thus $[L_1] \subseteq [L_2]$ implies $L_1 \subseteq [L_2]$. □

For any two ω-regular languages L_1, L_2 over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ recognised by NBA $\mathcal{A}$, $\mathcal{B}$ respectively, buffered simulation can be used to approximate the inclusion $L_1 \subseteq [L_2]$.

Theorem 5.2.8. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. If $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ for some $\kappa \in (\mathbb{N} \cup \{\omega\})^n$ then $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$.*

Algorithm 4 Checking $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$ for $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$.

```
1: $(k_1, \ldots, k_n) \leftarrow (0, \ldots, 0)$
2: $i \leftarrow 1$
3: **while** $\mathcal{A} \not\sqsubseteq^{k_1,\ldots,k_n} \mathcal{B}$ **do**
4:     $k_i \leftarrow k_i + 1$
5:     **If** $i < n$ **then** $i \leftarrow i + 1$ **else** $i \leftarrow 1$ **end if**
6: **end while**
7: **return** yes
```

Proof. Let $w \in L(\mathcal{A})$. Since w is accepted by $\mathcal{A}$, there is an accepting run $\rho \in \mathsf{AccRun}(\mathcal{A})$ over w. If Spoiler plays ρ and Duplicator plays according to the winning strategy then Duplicator forms an accepting run $\rho' \in \mathsf{AccRun}(\mathcal{B})$. Suppose $v = \mathsf{word}(\rho')$. Let $i \in \{1, \ldots, n\}$ and $\pi_i(w) = a_1 a_2 \ldots \in \Sigma_i^\infty$. We have $a_1, a_2, \ldots$ being the sequence of letters that are pushed by Spoiler consecutively to buffer i. Since Duplicator only pops letters that have been pushed to the buffers, $\pi_i(v)$ is a prefix of $a_1 a_2 \ldots$. If it is a proper prefix then there is some letter that is pushed to buffer i, but not popped by Duplicator. By definition of buffered simulation, Duplicator loses the play. This contradicts that Duplicator plays according to the winning strategy. Hence $\pi_i(v) = \pi_i(w)$. Since we consider an arbitrary i, this holds for all $i \in \{1, \ldots, n\}$. Hence $v \sim w$. Moreover since $v \in L(\mathcal{B})$, $w \in [L(\mathcal{B})]$. □

Buffered simulation with multiple buffers can be used to approximate trace closure inclusion incrementally in the same sense as buffered simulation with one buffer approximates language inclusion. Recall that by Theorem 3.5.4, buffered simulation with multiple buffers also admits a hierarchy. We have

$$\sqsubseteq^{\kappa_1} \subsetneq \sqsubseteq^{\kappa_2} \subsetneq \sqsubseteq^{\kappa_3} \ldots$$

for any capacity vectors $\kappa_1 < \kappa_2 < \ldots$. Duplicator gets stronger in showing trace closure inclusion if the capacity of the buffers grows. For any two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, we can check trace closure inclusion incrementally by starting with some small capacity vector $(k_1, \ldots, k_n) \in \mathbb{N}^n$ and check whether we have buffered simulation $\mathcal{A} \sqsubseteq^{k_1,\ldots,k_n} \mathcal{B}$. If this is the case then we conclude that we have trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$. Otherwise, we increase the capacity of one of the buffers and repeat the procedure again. We illustrate such a procedure in Algorithm 4.

Similar to the case of language inclusion, the incremental approximation for trace closure inclusion might not terminate. It does not terminate on any $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ where $[L(\mathcal{A})] \not\subseteq [L(\mathcal{B})]$ and also on any $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ where $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$, but $\mathcal{A} \not\sqsubseteq^{\kappa} \mathcal{B}$ for all $\kappa \in \mathbb{N}^n$.

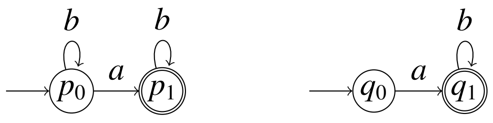
By Lemma 5.2.7 and Theorem 5.2.8, we have seen that buffered simulation implies trace closure inclusion. However, note that the converse does not hold. We have seen in Corollary 3.1.7 that language inclusion, a simple case of trace closure inclusion, does not imply buffered simulation. Hence one may ask whether there are cases, in which trace closure or language inclusion also implies buffered simulation. We will show that the answer to this question is related to the notion of continuity. If we have a continuous function that witnesses trace closure or language inclusion then trace closure or language inclusion also implies buffered simulation. We will show this in more detail in the following section.

5.3 Topological Characterisations

Before we give the topological characterisation of buffered simulation, we will show that the inclusion $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$, equivalently $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$, can be characterised by a function that maps the accepting runs of $\mathcal{A}$ to $\mathcal{B}$ over some trace equivalent words. Let us call such a function *trace-preserving*.

Definition 5.3.1. Let $\mathcal{A}, \mathcal{B}$ be two NBA over $\hat{\Sigma}$. A function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is called trace-preserving if for all $\rho \in \mathsf{AccRun}(\mathcal{A})$, $\mathsf{word}(\rho) \sim \mathsf{word}(f(\rho))$.

Example 5.3.2. Consider the following two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\{a\}, \{b\})$.



There is a unique trace-preserving function f, namely the one that maps every accepting run of $\mathcal{A}$ to a unique accepting run of $\mathcal{B}$, i.e. $f(\rho) = q_0 a (q_1 b)^\omega$ for all $\rho \in \mathsf{AccRun}(\mathcal{A})$. Such a function is trace-preserving since any accepting run of $\mathcal{A}$ is over some word $b^n a b^\omega$ where $n > 0$, and they are mapped to an accepting run of $\mathcal{B}$ over a trace equivalent word ab^ω.

If we have a trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, any word that is accepted by $\mathcal{A}$ has a trace equivalent word which is also accepted by $\mathcal{B}$. Hence we have $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$. On the other hand, if we have such an inclusion $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$ then it is clear that we can derive a trace-preserving function f.

Lemma 5.3.3. *$L(\mathcal{A}) \subseteq [L(\mathcal{B})]$ iff there exists a trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

Proof. Suppose there exists a trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$. Let $w \in L(\mathcal{A})$. Since w is accepted by $\mathcal{A}$, there is an accepting run $\rho \in \mathsf{AccRun}(\mathcal{A})$ over w. Let $\rho' = f(\rho)$ and $w' = \mathsf{word}(\rho')$. Since $\rho' \in \mathsf{AccRun}(\mathcal{B})$, we have $w' \in L(\mathcal{B})$, and since f is trace-preserving, we have $w \sim w'$. Hence by definition of the trace closure, $w \in [L(\mathcal{B})]$. Thus, $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$.

For the reverse direction, suppose $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$. For any $\rho \in \mathsf{AccRun}(\mathcal{A})$, we define $f(\rho)$ as the least accepting run in $\mathcal{B}$ with respect to the lexicographical order that satisfies $\mathsf{word}(f(\rho)) \sim \mathsf{word}(\rho)$. Such a run $f(\rho)$ exists since $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$. □

This shows that the inclusion $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$ can be characterised by the existence of a trace-preserving function. We will further see that such a characterisation can be lifted for buffered simulation by requiring the function f to be continuous.

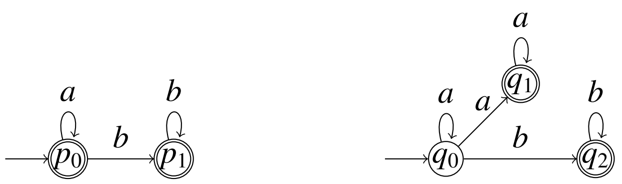
5.3.1 Characterisation of $\sqsubseteq^{\omega,\ldots,\omega}$

Throughout this section, we will define continuity of a trace-preserving function by considering the standard metric for infinite words [Tho90, PP04]. To show this, first for any automaton $\mathcal{A}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. An accepting run of $\mathcal{A}$ then can be seen as an infinite word over $Q^{\mathcal{A}} \cdot \Sigma$. The distance between two words is determined by the longest common prefix. If two words are the same then their distance is 0, and if their longest common prefix is of length $i - 1$ then their distance is $\frac{1}{2^i}$. The longer they share a common prefix, the closer is their distance.

Definition 5.3.4. Let Σ be an alphabet. The *distance* on Σ^ω is a function $d : \Sigma^\omega \times \Sigma^\omega \to [0, 1]$ where for every $w = a_0a_1 \ldots, w' = b_0b_1 \ldots, \in \Sigma^\omega$,

$$d(w, w') = \begin{cases} 0 & \text{if } w = w' \\ 2^{-i} & \text{if } a_i \neq b_i \text{ and for all } j < i,\ a_j = b_j. \end{cases}$$

Example 5.3.5. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\{a, b\})$.



Let $n \in \mathbb{N}$ be some number and $\rho_\mathcal{A}, \rho_\mathcal{B}$ be the accepting runs over a^nb^ω in $\mathcal{A}$, $\mathcal{B}$ respectively. There are such unique $\rho_\mathcal{A}$ and $\rho_\mathcal{B}$, i.e. $\rho_\mathcal{A} = (p_0a)^np_0b(p_1b)^\omega$ and $\rho_\mathcal{B} = (q_0a)^nq_0b(q_1b)^\omega$. Let $\rho'_\mathcal{A} = (p_0a)^\omega$ and $\rho'_\mathcal{B} = (q_0a)(q_1a)^\omega$ be two accepting runs over a^ω in $\mathcal{A}$ and $\mathcal{B}$ respectively. The distance between $\rho_\mathcal{A}$ and $\rho'_\mathcal{A}$ is $2^{-(n+1)}$ and the distance between $\rho_\mathcal{B}$ and $\rho'_\mathcal{B}$ is 2^{-2}.

It is not hard to see that the distance function given in Definition 5.3.4 is a metric. We will use it to determine the continuity of a trace-preserving function. Recall that from the standard definition in topology, a function is continuous if for any two inputs that are very close, their outputs are also very close. Hence for a function over the accepting runs of $\mathcal{A}$, $\mathcal{B}$, continuity is defined as follows.

Definition 5.3.6. A function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is *continuous* if for every $\rho \in \mathsf{AccRun}(\mathcal{A})$ and $i \in \mathbb{N}$, there exists $j \in \mathbb{N}$ such that for all $\rho' \in \mathsf{AccRun}(\mathcal{A})$,

$$d(\rho, \rho') < 2^{-j} \Rightarrow d(f(\rho), f(\rho')) < 2^{-i}.$$

If a function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is continuous, this intuitively means that for any two distinct accepting runs in $\mathcal{A}$ that share a very long common prefix, their images in $\mathcal{B}$ also do not diverge very early. Thus each of the transitions in the output run can be determined by looking at some finite prefix of the input run. If we need to look at the whole input run then there are two very similar accepting runs in which their images already differ in some early transition. In the following, we give an example of a non-continuous trace-preserving function.

Example 5.3.7. Consider again the NBA $\mathcal{A}$, $\mathcal{B}$ from Example 5.3.5. Let $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ be a trace-preserving function. Hence there exists an $n_0 > 0$ such that the accepting run ρ over a^ω in $\mathcal{A}$ is mapped into

$$f(\rho) = (q_0a)^{n_0}(q_1a)^\omega.$$

Such a function f is not continuous. To show this, for any $n \in \mathbb{N}$, let ρ_n be the accepting run in $\mathcal{A}$ over a^nb^ω. By Example 5.3.5, we know that there is such a unique ρ_n for each $n \in \mathbb{N}$. Moreover let $i_0 = n_0 + 1$. Hence for all $n \in \mathbb{N}$, we have $d(\rho, \rho_n) = 2^{-(n+1)}$, but $d(f(\rho), f(\rho_n)) = 2^{-(n_0+1)} \nless 2^{-i_0}$. By Definition 5.3.6, f is not continuous.

If we have buffered simulation $\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B}$ then we can construct a continuous trace-preserving function f from the winning strategy of Duplicator in the game $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A},\mathcal{B})$. For any accepting run ρ in $\mathcal{A}$, we define $f(\rho)$, as the run that is formed by Duplicator in $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A},\mathcal{B})$, assuming that Spoiler plays ρ and Duplicator plays according to the winning strategy. Such a function is continuous since each of the transitions of the output run can be determined by looking at some finite prefix of the input run. We show this formally as follows.

Lemma 5.3.8. *$\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B}$ only if there exists a continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

Proof. Let f be the function as we have defined before. To show that f is continuous, let ρ be an accepting run in $\mathcal{A}$ and $i \in \mathbb{N}$ a number. If Spoiler plays ρ in the game $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A},\mathcal{B})$ and Duplicator plays according to the winning strategy then there is a round $j > 0$ such that Duplicator forms a finite run of length i. Let ρ' be an accepting run in $\mathcal{A}$ that is distinct from ρ, but shares with ρ a common prefix of length at least j, i.e. $d(\rho,\rho') < 2^{-j}$. In the first j rounds, Duplicator does not see any difference whether Spoiler is actually playing ρ or ρ'. Duplicator makes the same moves in response to ρ or ρ'. This implies that the output runs $f(\rho)$ and $f(\rho')$ share the same prefix of length i. Hence

$$d(f(\rho), f(\rho')) < 2^{-i}. \tag{5.2}$$

Thus we have (5.2) if $d(\rho,\rho') < 2^{-j}$. Since we consider an arbitrary ρ and i, this hold for all $\rho \in \mathsf{AccRun}(\mathcal{A})$ and $i \in \mathbb{N}$. By definition, the function f is continuous. □

The reverse direction of this lemma also holds. We will show this by using an intermediate game called Γ-*game*, which is inspired from the delay simulation game [HKT10]. The winning condition is determined by some set of pairs of words that are produced by Spoiler and Duplicator.

Γ-Game

A Γ-game is a two-player game that is played similarly as the buffered simulation game. It is played on two NBA $\mathcal{A}$, $\mathcal{B}$ with a delay $k \in \mathbb{N}$. The winning condition however is not determined by the acceptance of $\mathcal{A}$ and $\mathcal{B}$, but by an objective function $f : R_1 \to R_2$ where $R_1 \subseteq \mathsf{Run}(\mathcal{A})$ and $R_2 \subseteq \mathsf{Run}(\mathcal{B})$. Intuitively, in every round, Spoiler moves the pebble in $\mathcal{A}$ by reading a letter and Duplicator either skips her turn or moves the pebble in $\mathcal{B}$ by reading a word. The length difference between Spoiler's and Duplicator's runs at the end of the round is at most k. The main difference to the buffered simulation game is that, in the Γ-game, the letters that are read by Spoiler are not put to buffers. Duplicator can read any word freely even using letters that are not yet read by Spoiler. This is not the case in the buffered simulation game. If one of the players gets stuck then the opponent wins. Otherwise the play goes on infinitely many rounds and produces two infinite runs ρ, ρ' of $\mathcal{A}$, $\mathcal{B}$, respectively. Duplicator wins iff whenever $\rho \in \mathsf{Dom}(f)$ then $\rho' = f(\rho)$. We formally define the game as follows.

Definition 5.3.9. Given two NBA $\mathcal{A}, \mathcal{B}$, a delay $k \in \mathbb{N}$, and an objective function $f : R_1 \to R_2$ where $R_1 \subseteq \mathsf{Run}(\mathcal{A})$ and $R_2 \subseteq \mathsf{Run}(\mathcal{B})$, the Γ-game $\Gamma^k(\mathcal{A},\mathcal{B},f)$ is defined as $((V, V_0, V_1, E), v_0, \mathsf{Win})$ where Spoiler's and Duplicator's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times \{0,\dots,k\} \times (\Sigma \cdot Q^{\mathcal{B}})^{\le k+1} \cup Q^{\mathcal{B}} \times \{\mathsf{S}\},$$

$$V_0 = \Sigma \cdot Q^{\mathcal{A}} \times \{1, \ldots, k+1\} \times Q^{\mathcal{B}} \times \{\mathsf{D}\},$$

$V = V_0 \cup V_1$, the edge relation E is defined as

$$(p, i, a_1 q_1 \ldots a_m q_m, \mathsf{S}) \to (ap', i+1, q_m, \mathsf{D}) \text{ in } E \qquad \text{iff} \qquad p \xrightarrow{a} p',$$

$$(ap, i, q, \mathsf{D}) \to (p, i-m, a_1 q_1 \ldots a_m q_m, \mathsf{S}) \text{ in } E \qquad \text{iff} \qquad q \xrightarrow{a_1} q_1 \ldots \xrightarrow{a_m} q_m,$$

the initial configuration is $v_0 = (p_0, 0, q_0, \mathsf{S})$ where p_0, q_0 is the pair of the initial states of $\mathcal{A}$, $\mathcal{B}$, and an infinite play

$$(r_0, i_0, s_0, \mathsf{S})(r_1, i'_1, s_0, \mathsf{D})(r_1, i_1, s_1, \mathsf{S})(r_2, i'_2, s_1, \mathsf{D})(r_2, i_2, s_2, \mathsf{S}) \ldots$$

is in Win iff $\rho = r_0 r_1 r_2 \ldots \notin R_1$ or $f(\rho) = s_0 s_1 s_2 \ldots$.

Intuitively, the first and the third components in the configuration remember the last move that is made by Spoiler and Duplicator. Hence collecting the first and the third components of the configurations in a play gives us the infinite runs that are formed by Spoiler and Duplicator, respectively. The second component remembers the length difference between Spoiler's and Duplicator's runs.

Example 5.3.10. Consider the two NBA $\mathcal{A}$, $\mathcal{B}$ and a trace-preserving function $f : \mathsf{Acc Run}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ as in Example 5.3.2. Duplicator wins the Γ-game $\Gamma^0(\mathcal{A}, \mathcal{B}, f)$ with the following winning strategy. Initially, Spoiler will read a or b by moving his pebble from p_0 to p_0 or p_1, respectively. He proceeds from the initial configuration $(p_0, 0, q_0, \mathsf{S})$ to $(bp_0, 1, q_0, \mathsf{D})$ or $(ap_1, q_0, 1, \mathsf{D})$. Duplicator reads a and moves her pebble to q_1. She proceeds to $(p_0, 0, aq_1, \mathsf{S})$ or $(p_1, 0, aq_1, \mathsf{S})$ accordingly. Spoiler again will continue by reading a or b and moves his pebble to p_0 or p_1. He proceeds to

$$(xp, 1, q_1, \mathsf{D}) \tag{5.3}$$

where $x \in \{a, b\}$ and $p \in \{p_0, p_1\}$. From such a configuration, Duplicator continues by reading b and loops in q_1 for the rest of the play, i.e. she proceeds to $(p, 0, bq_1, \mathsf{S})$.

Hence for every accepting run ρ that is formed by Spoiler, Duplicator forms the run $\rho' = p_0 a (q_1 b)^\omega$ which is nonetheless the f-image of ρ. Duplicator wins the game $\Gamma^0(\mathcal{A}, \mathcal{B}, f)$.

We can naturally extend the Γ-game to the case where we consider an unbounded delay, i.e. $k = \omega$. The definition of the game $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$ is the same as in Definition 5.3.9, but Spoiler's and Duplicator's configurations are respectively

$$V_1 = Q^{\mathcal{A}} \times \mathbb{N} \times (\Sigma \cdot Q^{\mathcal{B}})^* \cup Q^{\mathcal{B}} \times \{\mathsf{S}\} \text{ and}$$
$$V_0 = \Sigma \cdot Q^{\mathcal{A}} \times \mathbb{N}^+ \times Q^{\mathcal{B}} \times \{\mathsf{D}\}.$$

In [HKT10], for any ω-regular languange $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, it is shown that Duplicator wins the delay simulation game $\mathcal{G}^{f'}_{\mathsf{Del}}(L)$ with some delay function f' iff there is a continuous function $\lambda : \Sigma_1^\omega \to \Sigma_2^\omega$ that satisfies $(a_1, b_1)(a_2, b_2) \ldots \in L$ if $\lambda(a_1 a_2 \ldots) = b_1 b_2 \ldots$. We can show that a similar property holds for the Γ-game. The the function λ intuitively corresponds to the objective function in the Γ-game. Duplicator wins the Γ-game whenever the objective function f is continuous. We can derive a winning strategy from such a continuous function. Duplicator simply waits until she reaches a round where for every infinite run that can be extended from Spoiler's current run, its f-image can be extended from Duplicator's run. The continuity of f guarantees that Duplicator eventually will move.

Lemma 5.3.11. Duplicator *wins* $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$ *if f is continuous.*

Proof. The winning strategy for Duplicator is as follows. Suppose Spoiler and Duplicator have formed the runs $r_\mathcal{A}$ and $r_\mathcal{B}$ respectively. If Spoiler extends his run one step to $r'_\mathcal{A} = r_\mathcal{A}ap$ then Duplicator moves if for every run $\rho \in \mathsf{Dom}(f)$ that can be extended from $r'_\mathcal{A}$, the image $f(\rho)$ can be extended from $r'_\mathcal{B} = r_\mathcal{B}b_1q_1 \ldots b_nq_n$ for some $n > 0$. In such a case, Duplicator extends her run to such a maximal $r'_\mathcal{B}$. Otherwise, Duplicator skips her turn.

Duplicator will always form an infinite run whenever Spoiler forms some infinite run from $\mathsf{Dom}(f)$. We can show this by contradiction. Suppose Spoiler forms an infinite run $\rho \in \mathsf{Dom}(f)$, but after some round $m \in \mathbb{N}$, Duplicator always skips her turn. Let $r_\mathcal{A}, r_\mathcal{B}$ be the runs that are formed by Spoiler and Duplicator respectively, in round m, and for any $i \in \mathbb{N}$, let $r^i_\mathcal{A} = r_\mathcal{A}a_1q_1 \ldots a_iq_i$ be the run that is formed by Spoiler in round $m + i$. According to the winning strategy, for all $i \in \mathbb{N}$, there exists $\rho_i \in \mathsf{Dom}(f)$ that is distinct from ρ such that ρ and ρ_i can be extended from $r^i_\mathcal{A}$ but their images: $f(\rho)$ and $f(\rho_i)$ cannot be extended from some $r'_\mathcal{B} = r_\mathcal{B}a_1q_1$, an extension of $r_\mathcal{B}$. Otherwise, Duplicator would have extended her run after round m. Hence for all $i > 0$, we have

$$d(\rho, \rho_i) \leq 2^{-(|r_\mathcal{A}|+1+i)} \text{ and}$$
$$d(f(\rho), f(\rho_i)) = 2^{-(|r_\mathcal{B}|+1)}.$$

In other words, the sequence $\rho_1, \rho_2, \ldots$ gets closer to ρ but $f(\rho_1)$, $f(\rho_2)$, $\ldots$ differ from ρ already in the $(|r_\mathcal{B}| + 1)$-th transition. This contradicts that f is continuous. Hence if Spoiler forms an infinite run $\rho \in \mathsf{Dom}(f)$, Duplicator also forms an infinite run ρ'.

Moreover note that the following invariant holds: in any round i where Spoiler's and Duplicator's runs are respectively $r^{(i)}_\mathcal{A}$ and $r^{(i)}_\mathcal{B}$, if there is $\rho \in \mathsf{Dom}(f)$ that is started with $r^{(i)}_\mathcal{A}$ then $f(\rho)$ is started with $r^{(i)}_\mathcal{B}$. Hence whenever Spoiler forms a run ρ in $\mathsf{Dom}(f)$, Duplicator forms the f-image of ρ. Thus Duplicator wins $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$. □

The reverse direction of this lemma also holds. Duplicator loses any Γ-game $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$ if f is not continuous. The winning strategy for Spoiler is to play the run $\rho \in \mathsf{Dom}(f)$ in which f is not continuous at ρ, i.e. for all $i \in \mathbb{N}$, we have $\rho_i \in \mathsf{Dom}(f)$ that shares the same prefix of length i with ρ, but their images $f(\rho_i)$ and $f(\rho)$ differ already in some early transitions. We show this formally in the following lemma.

Lemma 5.3.12. Duplicator *wins* $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$ *iff f is continuous.*

Proof. The right-to-left direction holds because of Lemma 5.3.11. For the left-to-right direction, suppose f is not continuous. By definition, there exists $\rho \in \mathsf{Dom}(f)$ and $i_0 \in \mathbb{N}$ such that for every $i \in \mathbb{N}$, we have $\rho_i \in \mathsf{Dom}(f)$ that is distinct from ρ and $d(\rho, \rho_i) < 2^{-i}$, but $d(f(\rho), f(\rho_i)) \geq 2^{-i_0}$. The winning strategy for Spoiler is to first play ρ until Duplicator forms a finite run $r_\mathcal{B}$ of length i_0. Suppose this happens in round n. If $r_\mathcal{B}$ is not a prefix of $f(\rho)$ then Spoiler keeps playing ρ for the rest of the play. Otherwise, he continues by playing ρ_n. This is possible, since by definition, the runs ρ and ρ_n share the same prefix of length n. In the first case, Spoiler wins because Duplicator does not form the f-image of ρ. In the second case, since $d(f(\rho), f(\rho_n)) \geq 2^{-i_0}$, the output runs $f(\rho)$ and $f(\rho_n)$ share the same prefix of length at most $i_0 - 1$. Since $|r_\mathcal{B}| = i_0$ and $r_\mathcal{B}$ is a prefix of $f(\rho)$, $r_\mathcal{B}$ is not a prefix of $f(\rho_n)$. Hence Duplicator also does not form the f-image of ρ_n. Spoiler forms either the run ρ or $\rho_n \in \mathsf{Dom}(f)$, but Duplicator does not form the corresponding f-image of Spoiler's run. Hence Spoiler wins $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$. □

In the case where Duplicator wins the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ and f is trace-preserving then Duplicator also wins the buffered simulation game $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$. At a glance, it seems that Duplicator can use the winning strategy from the Γ-game for the buffered simulation game. Unfortunately this is not the case. The winning strategy in the Γ-game may not be suitable for the buffered simulation game. The reason is that the strategy in the Γ-game may tell Duplicator to output a letter that is not yet in the buffer.

Example 5.3.13. Consider again the two NBA $\mathcal{A}, \mathcal{B}$ from Example 5.3.2. It is shown that Duplicator wins the Γ-game $\Gamma^{0}(\mathcal{A}, \mathcal{B}, f)$ and hence she also wins the game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ with the same winning strategy. She wins with a winning strategy that tells Duplicator to read a in the first round and then b for the rest of the play. This strategy however is not suitable for the game $\mathcal{G}^{\omega,\omega}(\mathcal{A}, \mathcal{B})$ since Spoiler may not read a in the first round.

The fundamental reason why we cannot simply use the winning strategy in the Γ-game is because the move of Duplicator in the buffered simulation game is more restricted than in the Γ-game. In the buffered simulation game $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$ where $\mathcal{A}, \mathcal{B}$ are NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, if Duplicator pops $a_1, a_2, \ldots$ from buffer i then $a_1, a_2, \ldots$ is exactly the sequence of letters that have been put by Spoiler to buffer i. Hence if $v, w \in \Sigma^*$ are the words that are produced by Spoiler and Duplicator after some finite rounds in $\mathcal{G}^{\kappa}(\mathcal{A}, \mathcal{B})$, $\pi_i(w)$ is a prefix of $\pi_i(v)$ for all $i \in \{1, \ldots, n\}$. This however is not the case in the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ since Duplicator can produce any run independently of Spoiler's run.

A Sufficient and Necessary Condition for $\sqsubseteq^{\omega,\ldots,\omega}$

Nevertheless, for any two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a trace preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, we can translate Duplicator's winning strategy in the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ to the buffered simulation game $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$. In the buffered simulation game $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$, Duplicator simply skips her turn if the winning strategy in the Γ-game tells Duplicator to output a letter $a \in \Sigma$ that is not yet read by Spoiler. Since f is trace preserving, Spoiler eventually will read a. In such a case, Duplicator pops a and tries to form the same run that she would have formed in the Γ-game maximally. We show this formally as follows.

Lemma 5.3.14. *Let $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ be a trace-preserving function. If* Duplicator *wins $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ then she also wins $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$.*

Proof. Suppose the NBA $\mathcal{A}$, $\mathcal{B}$ are over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and Duplicator wins the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$. The winning strategy in the buffered simulation game $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$ is as follows. Suppose Spoiler and Duplicator currently have formed the runs $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$. Let $w = \mathsf{word}(r_{\mathcal{A}})$ and $w' = \mathsf{word}(r_{\mathcal{B}})$. If the strategy in the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ tells Duplicator to extend $r_{\mathcal{B}}$ to $r'_{\mathcal{B}} := r_{\mathcal{B}} b_1 p_1 \ldots b_m p_m$, $m \geq 0$, then in the buffered simulation game, we extend $r_{\mathcal{B}}$ to $r_{\mathcal{B}} b_1 q_1 \ldots b_{m'} q_{m'}$, $m' \leq m$, the maximal prefix of $r'_{\mathcal{B}}$ such that $\pi_i(w' b_1 \ldots b_{m'})$ is a prefix of $\pi_i(w)$ for all $i \in \{1, \ldots, n\}$.

Duplicator will form the same run that she would have formed in the game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$. Suppose Spoiler plays an accepting run ρ and let ρ_1, ρ_2 be the runs that are formed by Duplicator in $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$ and $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A}, \mathcal{B})$. Since we always extend Duplicator's run in the buffered simulation game by taking the prefix of the original extension in the Γ-game, it is clear that any finite prefix of ρ_2 is also a prefix of ρ_1. We will show that the converse also holds: any finite prefix of ρ_1 is a prefix of ρ_2. Suppose $r_{\mathcal{B}}$ is a finite prefix of ρ_1. Hence there is $r_{\mathcal{A}}$, a finite prefix of ρ, such that in the Γ-game, when Spoiler extends his run to $r_{\mathcal{A}}$, DUPLICATOR extends her run to $r_{\mathcal{B}}$. Let $w = \mathsf{word}(r_{\mathcal{A}})$ and $w' = \mathsf{word}(r_{\mathcal{B}})$. Since f is trace-preserving, there is $r'_{\mathcal{A}} := r_{\mathcal{A}} a_1 p_1 \dots a_k p_k$, $k \geq 0$, a finite prefix of ρ, such that $\pi_i(w')$ is a prefix of $\pi_i(wa_1 \dots a_k)$ for all $i \in \{1, \dots, n\}$. Hence when SPOILER forms $r'_{\mathcal{A}}$, DUPLICATOR extends her run to $r_{\mathcal{B}}$ in the Γ-game. This implies that $r_{\mathcal{B}}$ is a prefix of ρ_2. Hence we have $\rho_1 = \rho_2$.

Now suppose SPOILER plays an accepting run $\rho \in \mathsf{AccRun}(\mathcal{A})$ in $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A}, \mathcal{B})$. DUPLICATOR then forms $f(\rho)$. By definition of f, the run $f(\rho)$ is accepting. Hence DUPLICATOR also wins the play in $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A}, \mathcal{B})$. □

We can put this lemma together with Lemma 5.3.11 to show that the existence of a continuous trace-preserving function is indeed sufficient for buffered simulation.

Lemma 5.3.15. *$\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B}$ if there exists a continuous trace-preserving function f : $\mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

Proof. Suppose we have a continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$. Since f is continuous, by Lemma 5.3.11, DUPLICATOR wins the Γ-game $\Gamma^{\omega}(\mathcal{A}, \mathcal{B}, f)$. Moreover since f is trace-preserving, by Lemma 5.3.14, DUPLICATOR also wins the game $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A}, \mathcal{B})$. □

If we put Lemma 5.3.15 and Lemma 5.3.8 together, we obtain a characterisation of buffered simulation. This characterisation can be seen as a refinement of the characterisation of trace closure inclusion. Recall that in Lemma 5.3.3, we characterise trace closure inclusion by the existence of a trace-preserving function. For buffered simulation, such a function is required to be continuous.

Theorem 5.3.16. *$\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B}$ if and only if there exists a continuous trace-preserving function f : $\mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

By using this characterisation, we can classify pairs of automata in which their language or trace closure inclusion cannot be shown by buffered simulation. Let Incl be the set of all triples $\mathcal{A}$, $\mathcal{B}$, $\hat{\Sigma}$ where $\mathcal{A}$, $\mathcal{B}$ are NBA over $\hat{\Sigma}$ and we have $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$. Moreover, let $\mathsf{Cont} \subseteq \mathsf{Incl}$ such that a triple $\mathcal{A}$, $\mathcal{B}$, $\hat{\Sigma}$ is in Cont iff we have a continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, and $\mathsf{DisCont} = \mathsf{Incl} \setminus \mathsf{Cont}$. If the triple $\mathcal{A}$, $\mathcal{B}$, $\hat{\Sigma}$ is in Cont then by Lemma 5.3.3 and Theorem 5.3.16, $\mathcal{A}$, $\mathcal{B}$ are automata over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ where $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ for some $\kappa \in (\mathbb{N} \cup \{\omega\})^n$. On the other hand, if the triple $\mathcal{A}, \mathcal{B}, \hat{\Sigma}$ is in $\mathsf{DisCont}$ then $\mathcal{A} \not\sqsubseteq^{\kappa} \mathcal{B}$ for any $\kappa \in (\mathbb{N} \cup \{\omega\})^n$.

Example 5.3.17. Consider the NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\{a, b\})$ in Example 5.3.5. It is not hard to see that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ since their language are the same. We have a trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$. However in Example 5.3.7 we have seen that any of such function f is not continuous. Hence the triple $\mathcal{A}$, $\mathcal{B}$, $\hat{\Sigma}$ belongs to $\mathsf{DisCont}$. This implies that $\mathcal{A} \not\sqsubseteq^{k} \mathcal{B}$ for all $k \in \mathbb{N} \cup \{\omega\}$.

5.3.2 Characterisation of $\sqsubseteq^{k,\dots,k}$

Theorem 5.3.16 shows that we can characterise buffered simulation with unbounded buffers by the existence of a continuous trace-preserving function. One natural question then is to ask whether we can have a similar characterisation for buffered simulation with bounded buffers. Intuitively, we should be able to obtain such a characterisation by requiring the trace preserving function to be Lipschitz continuous instead of only continuous.

In [HKT10], for any ω-regular languange $L \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, it is shown that Duplicator wins the delay simulation game $\mathcal{G}^{f'}_{\mathsf{Del}}(L)$ with some constant delay function f' iff there is a Lipschitz continuous function $\lambda : \Sigma_1^\omega \to \Sigma_2^\omega$ that satisfies $(a_1, b_1)(a_2, b_2)\ldots \in L$ if $\lambda(a_1 a_2 \ldots) = b_1 b_2 \ldots$. We will show that we can also have a similar characterisation for buffered simulation. However we will see that such a characterisation only holds for some restricted cases.

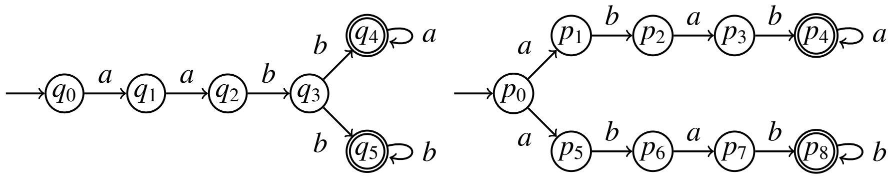
First recall that a function is Lipschitz continuous if there exists a constant c such that for any two inputs of distance d, their outputs' distance is at most $c \cdot d$.

Definition 5.3.18. A function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is Lipschitz continuous if there exists a constant $c \geq 0$ such that for any $\rho, \rho' \in \mathsf{AccRun}(\mathcal{A})$, we have

$$d(f(\rho), f(\rho')) \leq 2^c\, d(\rho, \rho').$$

Intuitively, if a function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is Lipschitz continuous with constant c then for any two distinct accepting runs ρ, ρ' in $\mathcal{A}$, if their outputs $f(\rho)$, $f(\rho')$ diverge at position i, the runs ρ and ρ' diverge already at position $(i + c)$. Thus we can determine the i-th transition of the output run by looking at the first $(i + c)$ transitions of the input run.

Example 5.3.19. Consider the two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\{a\}, \{b\})$ from Example 3.4.7. For convenience, we present them again as follows.



There is a unique trace-preserving function f, namely the one that maps the accepting runs over $aabba^\omega$ and aab^ω in $\mathcal{A}$ to the ones over $ababa^\omega$ and $abab^\omega$ in $\mathcal{B}$, respectively. The function f is Lipschitz continuous with constant $c \geq 3$. This is because for any two accepting runs ρ, ρ' in $\mathcal{A}$, the distance $d(\rho, \rho')$ is either 0 or 2^{-4} and the distance $d(f(\rho), f(\rho'))$ is either 0 or 2^{-1}. Hence $d(f(\rho), f(\rho')) \leq 2^c\, d(\rho, \rho')$.

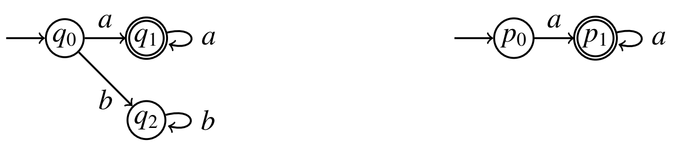
For any two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, if we have buffered simulation $\mathcal{A} \sqsubseteq^{k,\ldots,k} \mathcal{B}$ for some $k \in \mathbb{N}$ then we can construct a trace-preserving function f that is Lipschitz continuous with constant $c = nk$. We can construct such a function f from the winning strategy of Duplicator in the same way as in Lemma 5.3.8.

Lemma 5.3.20. *If $\mathcal{A} \sqsubseteq^{k,\ldots,k} \mathcal{B}$ for some $k \in \mathbb{N}$ then there exists a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

Proof. For every accepting run ρ in $\mathcal{A}$, we define $f(\rho)$ as the accepting run that is formed by Duplicator in the game $\mathcal{G}^{k,\ldots,k}(\mathcal{A}, \mathcal{B})$, assuming that Spoiler plays ρ and Duplicator plays according to the winning strategy. Such a function is Lipschitz continuous with constant $c = nk$, which is the total capacity of all the buffers. To show this, let ρ be an accepting run in $\mathcal{A}$ and $n \in \mathbb{N}$ be a number. If Spoiler plays ρ then in round $n + c$, Duplicator forms a run of length at least n. Let ρ' be an accepting run in $\mathcal{A}$ that is distinct from ρ, but shares the same prefix of length $n + c$ with ρ. In the first $n + c$ rounds, Duplicator does not see any difference whether Spoiler is actually playing ρ or ρ'. Duplicator makes the same move in response to ρ or ρ'. This implies that $d(f(\rho), f(\rho')) \leq 2^c \cdot d(\rho, \rho')$. Hence by definition, f is Lipschitz continuous with constant c. □

The reverse direction of this lemma however does not hold. There is a case where we have a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^{k,\dots,k} \mathcal{B}$ for any $k \in \mathbb{N}$. One simple reason is that Spoiler can play a non-accepting run in $\mathcal{A}$ that cannot be mimicked by Duplicator.

Example 5.3.21. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\{a, b\})$.

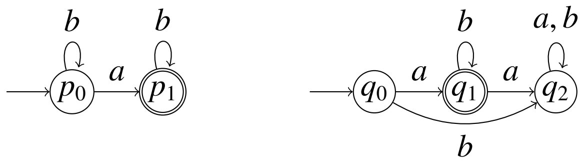


There is a unique Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$. It is $f(\rho) = p_0a(p_1a)^\omega$ for all $\rho \in \mathsf{AccRun}(\mathcal{A})$. This function is trace-preserving since there is only one accepting run in $\mathcal{A}$, which is over a^ω, and it is mapped to a run over the same word. The function f is Lipschitz continuous with Lipschitz constant 0 since there is only one output run. Hence the distance between two output runs is always 0. However we have $\mathcal{A} \not\sqsubseteq^k \mathcal{B}$ for all $k \in \mathbb{N}$. Spoiler wins the game $\mathcal{G}^k(\mathcal{A}, \mathcal{B})$ by playing the word b^ω. Duplicator eventually fills the buffer more than its capacity and hence loses the play.

This example shows that in the game with bounded buffers, Duplicator might lose the play even though Spoiler does not form an accepting run. This however is irrelevant to the use of buffered simulation as an approximation for trace closure inclusion. We are only interested in accepting runs that can be formed by Spoiler. We can simply avoid this by assuming that Duplicator's automaton is complete. Without loss of generality, we will assume that for every $q \in Q^\mathcal{B}$ and $a \in \Sigma$, there exists $q' \in Q^\mathcal{B}$ such that $(q, a, q') \in E^\mathcal{B}$. If Duplicator's automaton is complete then she can mimic any non-accepting run that is formed by Spoiler.

However, the reason why Lipschitz continuity is not enough to capture buffered simulation with bounded buffers is more complex. Even if we assume that Duplicator's automaton is complete, there are still cases in which we have a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, but $\mathcal{A} \not\sqsubseteq^{k,\dots,k} \mathcal{B}$ for all $k \in \mathbb{N}$.

Example 5.3.22. Consider the following two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\{a\}, \{b\})$.



In this case, Duplicator's automaton $\mathcal{B}$ is complete. Moreover we also have a unique Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$, namely the one that maps every accepting run in $\mathcal{A}$ to the one over ab^ω in $\mathcal{B}$, i.e. $f(\rho) = q_0a(q_1b)^\omega$ for all $\rho \in \mathsf{AccRun}(\mathcal{A})$. The function f is trace-preserving since any accepting run in $\mathcal{A}$ is over the word b^nab^ω, $n \geq 0$, and it is mapped to an accepting run over trace equivalent word: ab^ω. It is also Lipschitz continuous since the function maps every accepting run in $\mathcal{A}$ to the same run in $\mathcal{B}$ and therefore the distance between two output runs is always 0. Hence the function f is Lipschitz continuous with Lipschitz constant 0. However we have $\mathcal{A} \not\sqsubseteq^{k,k} \mathcal{B}$, for any $k \in \mathbb{N}$. Spoiler wins the game $\mathcal{G}^{k,k}(\mathcal{A}, \mathcal{B})$ by first reading $bbb\dots$

indefinitely. Duplicator eventually moves to the non-accepting sink q_2, i.e. she proceeds to the configuration $(p_0, (\epsilon, w), q_2, \bot, \mathsf{S})$ where $w \in b^{\leq k}$. From this configuration, Spoiler can continue by reading ab^{ω} and form an accepting run. Duplicator will not form an accepting run from q_2 and hence lose the play.

In this example, the reason why Duplicator loses is different from Example 5.3.21. For every accepting run that can be formed by Spoiler in $\mathcal{A}$, there is a corresponding accepting run in $\mathcal{B}$ over a trace equivalent word. There is even a unique correspondence. Duplicator does not need any lookahead. She simply produces the accepting run over ab^{ω} no matter what Spoiler reads. However, to execute her first move, Duplicator has to wait until Spoiler reads a. Spoiler eventually will read a since otherwise he will lose for not producing an accepting run. However the problem is that there is no bound on how long Duplicator has to wait until Spoiler reads a. She first needs to store unboundedly many bs to the buffer. The main reason is because in the word that is produced by Spoiler, the letter a can commute unboundedly to the left or to the right. Spoiler may read a in a very late or early round, but Duplicator has to read a for her first move. To avoid this situation, we need to restrict Spoiler's automaton such that he only produces words where each letter cannot commute unboundedly. If each letter in the word produced by Spoiler only commutes at most d steps then whenever Duplicator knows she has to read a letter $a \in \Sigma$, the letter a will be read by Spoiler at most in the next d rounds. We will formalise such an automaton by considering the notion of *correspondence relation* and *commutative degree*.

The Correspondence Relation

First let us define a *correspondence relation* Corr that is defined for two given words $v, w \in \Sigma^*$. It relates the positions in v to the ones in w which carry the same letter and have the same order with respect to this letter.

Definition 5.3.23. For any $v, w \in \Sigma^*$, let $\mathsf{Corr}_{v,w} \subseteq \mathsf{Pos}(v) \times \mathsf{Pos}(w)$ such that $(i, i') \in \mathsf{Corr}_{v,w}$ iff

- $v(i) = w(i') = a$ for some $a \in \Sigma$, and
- $|v(1) \dots v(i)|_a = |w(1) \dots w(i')|_a$

Example 5.3.24. Consider the words $v = abbc$ and $w = acbb$ over $\Sigma = \{a, b, c\}$. In this case, we have $\mathsf{Corr}_{v,w} = \{(1, 1), (2, 3), (3, 4), (4, 1)\}$.

Intuitively, the position i in v corresponds to the position j in w if they are the positions of some letter $a \in \Sigma$ and both i and j are the positions of the n-th a. If v and w are trace equivalent with respect to some $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ then for any letter $a \in \Sigma_1 \cup \dots \cup \Sigma_n$, the number of occurrences of a in v and w are always the same. In such a case, a correspondence relation is nonetheless a bijection, and we will simply denote the corresponding position of $i \in \mathsf{Pos}(v)$ with $\mathsf{Corr}_{v,w}(i)$.

Now let $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ be some distributed alphabet, (Σ, D) its corresponding dependence alphabet, and $w \in \Sigma^*$. The correspondence relation can be used to characterise the letters in w that are in the dependence relation D.

Lemma 5.3.25. *If $(w(i), w(j)) \in D$ and $i \leq j$ then for any $v \sim w$,*

$$\mathsf{Corr}_{w,v}(i) \leq \mathsf{Corr}_{w,v}(j).$$

Proof. Let $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ be the corresponding distributed alphabet. We can show this by contradiction. Suppose $\mathsf{Corr}_{w,v}(i) > \mathsf{Corr}_{w,v}(j)$. Let $i' = \mathsf{Corr}_{w,v}(i)$ and $j' = \mathsf{Corr}_{w,v}(j)$. Moreover, let a, $b \in \Sigma$ such that $w(i) = v(i') = a$ and $w(j) = v(j') = b$. By definition of the correspondence relation,

$$|w(1)\dots w(i)|_a = |v(1)\dots v(i')|_a = n_1$$
$$|w(1)\dots w(j)|_{\mathsf{b}} = |v(1)\dots v(j')|_{\mathsf{b}} = n_2$$

for some $n_1, n_2 \in \mathbb{N}$. Since $(a, b) \in D$, there exists $k \in \{1, \dots, n\}$ such that $a, b \in \Sigma_k$. Since $i \leq j$, there exists at least n_1 many as before the n_2-th b in $\pi_k(w)$. Since $i' > j'$, there exists less than n_1 many as before the n_2-th b in $\pi_k(v)$. Hence $\pi_k(w) \neq \pi_k(v)$. This contradicts that $w \sim v$. Thus $i' \leq j'$. □

Intuitively, this lemma shows that if two letters of w at positions i and j are dependent on each other, their corresponding letters at positions i' and j' in any word $v \sim w$ never change order, i.e. if $i \leq j$ then $i' \leq j'$.

On the other hand, for any position $i_0 \in \mathsf{Pos}(w)$, if the letters at positions i_0 and $i_0 + 1$ are independent of each other, we can find a trace equivalent word v such that the positions i_0, $i_0 + 1$ in w correspond to the positions $i_0 + 1$, i_0 in v, respectively. For example, we can simply obtain v from w by swapping the letters at positions i_0 and $i_0 + 1$. We can generalise this property as follows.

Lemma 5.3.26. *If* $(w(i_0), w(i_0+1)), (w(i_0), w(i_0+2)), \dots, (w(i_0), w(i_0+m)) \notin D$ *then there exists* $v \sim w$ *such that*

$$\mathsf{Corr}_{w,v}(i) = \begin{cases} i + m & \text{if } i = i_0, \\ i - 1 & \text{if } i_0 < i \leq i_0 + m, \\ i & \text{otherwise.} \end{cases}$$

Proof. Let $\ell = |w|$ and $b_0, \dots, b_m \in \Sigma$ such that $w(i_0) = b_0$, $w(i_0 + 1) = b_1$, ..., $w(i_0 + m) = b_m$. Moreover let $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ be the corresponding distributed alphabet and

$$\begin{aligned} v_1 &= w(1)\dots w(i_0 - 1), \\ v_2 &= b_0\, b_1 \dots b_m, \\ v_2' &= b_1 \dots b_m\, b_0, \\ v_3 &= w(i_0 + (m+1))\dots w(\ell). \end{aligned}$$

Since $(b_0, b_1), \dots, (b_0, b_m) \notin D$, there is no $k \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$ such that $b_0, b_j \in \Sigma_k$. Hence for all $i \in \{1, \dots, n\}$, we have $\pi_i(v_2) = \pi_i(v_2') = b_0$ if $b_0 \in \Sigma_i$ and $\pi_i(v_2) = \pi_i(v_2') = \pi_i(b_1 \dots b_m)$ if $b_0 \notin \Sigma_i$. Thus $\pi_i(v_2) = \pi_i(v_2')$ for all $i \in \{1, \dots, n\}$. We have $v_2 \sim v_2'$. Let $v = v_1 v_2' v_3$. Since $w = v_1 v_2 v_3$ and $v_2 \sim v_2'$, it is clear that $v \sim w$.

By the construction of v, we have $w(i_0) = v(i_0{+}m) = b_0$. We also have $|w(1)\dots w(i_0)|_{b_0} = |v(1)\dots v(i_0 + m)|_{b_0}$ since $b_1, \dots, b_m$ are distinct from b_0. Hence $(i_0, i_0 + m) \in \mathsf{Corr}_{w,v}$.

For any $j \in \{1, \dots, m\}$, we have $w(i_0 + j) = v((i_0 + j) - 1) = b_j$. We also have $|w(1)\dots w(i_0 + j)|_{b_j} = |v(1)\dots v(i_0 + j)|_{b_j}$ since $b_1, \dots, b_m$ are distinct from b_0. Hence $(i_0 + 1, i_0), \dots, (i_0 + m, i_0 + (m-1)) \in \mathsf{Corr} w, v$.

For any $i \in \mathsf{Pos}(w)$ where $i < i_0$ and $i > i_0{+}m$, by the construction of v, we have $w(i) = v(i) = a$ for some $a \in \Sigma$ and $|w(1)\dots w(i)|_a = |v(1)\dots v(i)|_a$. Hence $(i, i) \in \mathsf{Corr}_{w,v}$. □

Intuitively, this lemma shows that if the letter at position i_0 is independent of the ones at position $i_0 + 1, \ldots, i_0 + m$, there exists a word v that is trace equivalent to w such that the position i_0 in w corresponds to the position $i_0 + m$ in v, and the positions $i_0 + 1, \ldots, i_0 + m$ in w correspond to the positions $i_0, \ldots, i_0 + m - 1$ in v, respectively. We simply can obtain such a word v from w by moving the letter at position i_0 over the ones at positions $i_0 + 1$, …, i_m.

The Commutative Degree of an Automaton $\mathcal{A}$

Now consider a word $w \in \Sigma^*$ over some distributed alphabet $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. The correspondence relation $\mathsf{Corr}_{w,v}$, where $v \sim w$, simply determines how far the letters in w can commute to the left or right. If there is a word $v \sim w$ where $(i, i + m) \in \mathsf{Corr}_{w,v}$ then the letter at position i in w can commute m steps to the right. Likewise for direction left. If there is a word $v' \sim w$ where $(i, i - m) \in \mathsf{Corr}_{w,v'}$ then the letter at position i in w can commute m steps to the left. Thus the maximal length of how far the letter in w can commute to the left or right can be determined by looking at all the correspondence relations $\mathsf{Corr}_{w,v}$ where $v \sim w$. Let us call such a maximal length *commutative degree*.

Definition 5.3.27. Let $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ be a distributed alphabet, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ the union of the alphabets, and $w \in \Sigma^*$. Let

$$\mathsf{Deg}^+_w(i) = \max\{j - i \mid (i, j) \in \mathsf{Corr}_{w,v},\ w \sim v,\ v \in \Sigma^*\}$$
$$\mathsf{Deg}^-_w(i) = \max\{i - j \mid (i, j) \in \mathsf{Corr}_{w,v},\ w \sim v,\ v \in \Sigma^*\}.$$

The commutative degree of the letter at position $i \in \mathsf{Pos}(w)$ is

$$\mathsf{Deg}_w(i) = \max\{\mathsf{Deg}^+_w(i), -\mathsf{Deg}^+_w(i)\}.$$

Intuitively $\mathsf{Deg}^+_w(i)$ is the maximal length of how far the letter at position i in w can commute to the right and $\mathsf{Deg}^-_w(i)$ is the maximal length of how far the letter at position i in w can commute to the left.

Example 5.3.28. Let $\hat{\Sigma} = (\{a, b\}, \{a, c\})$. Consider the word $w = acbb$. The set of all words that are trace equivalent to w is $S = \{acbb, abcb, abbc\}$. We have $\mathsf{Deg}_w(1) = 0$ since for all $v \in S$, $\mathsf{Corr}_{w,v}(1) = 1$. The first letter a cannot commute to the left or right. Moreover, we have $\mathsf{Deg}_w(2) = 2$ since

$$\mathsf{Corr}_{w,v}(2) = \begin{cases} 2 & \text{if } v = acbb \\ 3 & \text{if } v = abcb \\ 4 & \text{if } v = abbc. \end{cases}$$

Thus the letter c at position 2 can commute at most two steps to the right. We also have $\mathsf{Deg}_w(3) = \mathsf{Deg}_w(4) = 1$ since

$$\mathsf{Corr}_{w,v}(3) = \begin{cases} 3 & \text{if } v = acbb \\ 2 & \text{if } v = abcb \\ 3 & \text{if } v = abbc \end{cases} \qquad \text{and} \qquad \mathsf{Corr}_{w,v}(4) = \begin{cases} 4 & \text{if } v = acbb \\ 4 & \text{if } v = abcb \\ 3 & \text{if } v = abbc. \end{cases}$$

Thus both the letters b in w can commute at most one step to the left.

For any word $w \in \Sigma^*$, we define the commutative degree of w as the maximal commutative degree of its letters, i.e.

$$\mathsf{Deg}(w) = \max\{\mathsf{Deg}_w(i) \mid i \in \mathsf{Pos}(w)\}.$$

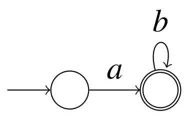
Hence $\mathsf{Deg}(w)$ is at most $|w| - 1$. For an automaton $\mathcal{A}$, we define the commutative degree of $\mathcal{A}$ as the supremum of the commutative degree of all finite words produced by $\mathcal{A}$, i.e.

$$\mathsf{Deg}(\mathcal{A}) = \sup\{\mathsf{Deg}(a_1 \dots a_n) \mid p_1 \xrightarrow{a_1} p_2 \dots \xrightarrow{a_n} p_{n+1}\}.$$

Hence we have $\mathsf{Deg}(\mathcal{A}) = k$ for some $k \in \mathbb{N}$ if for every word w over some finite run in $\mathcal{A}$, $\mathsf{Deg}_w(i) \leq k$ for all $i \in \mathsf{Pos}(w)$. In other words, each letter in the finite word produced by $\mathcal{A}$ commute at most k steps to the left or right. In such a case, let us call the commutative degree of $\mathcal{A}$ finite.

On the other hand, we have $\mathsf{Deg}(L) = \infty$ if for each $k \in \mathbb{N}$, there is a finite word w produced by $\mathcal{A}$ and a position i in w such that $\mathsf{Deg}_w(i) > k$. In this case, let us call the commutative degree of $\mathcal{A}$ infinite.

Example 5.3.29. Consider the following NBA $\mathcal{A}$ over $\hat{\Sigma} = (\{a\}, \{b\})$.



In this case, the commutative degree of $\mathcal{A}$ is infinite. To show this, let $k \in \mathbb{N}$ be some arbitrary number. Consider the word $w = ab^{k+1}$. The first letter of w commutes $k + 1$ steps to the right: there is a word $v = b^{k+1}a$ where $v \sim w$ and $(1, k + 2) \in \mathsf{Corr}_{w,v}$. Hence $\mathsf{Deg}_{w_k}(1) > k$. Since we consider an arbitrary k, this holds for any $k \in \mathbb{N}$. For all $k \in \mathbb{N}$, there exists a word w over some finite run in $\mathcal{A}$ such that $\mathsf{Deg}(w) > k$. Hence by definition, $\mathsf{Deg}(\mathcal{A}) = \infty$.

We will show that for any two NBA $\mathcal{A}$, $\mathcal{B}$, if the commutative degree of $\mathcal{A}$ is finite and $\mathcal{B}$ is complete then the reverse direction of Lemma 5.3.8 holds. We will show this by using the Γ-game that is defined in the previous subsection.

A Sufficient Condition for $\sqsubseteq^{k,\dots,k}$

First note that if Duplicator wins the Γ-game $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ where $k \in \mathbb{N}$ and $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is trace-preserving, this does not necessarily imply that Duplicator also wins the buffered simulation game $\mathcal{G}^{k,\dots,k}(\mathcal{A}, \mathcal{B})$. Consider again the two automata $\mathcal{A}$, $\mathcal{B}$ from Example 5.3.21 and a trace-preserving function $f(\rho) = q_0a(q_1a)^\omega$ for all $\rho \in \mathsf{AccRun}(\mathcal{B})$. Duplicator wins the Γ-game $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ with delay $k = 0$. She just reads a^ω no matter what Spoiler plays. However, Duplicator loses the buffered simulation game $\mathcal{G}^{0,0}(\mathcal{A}, \mathcal{B})$. Spoiler wins by reading b in the first round. Duplicator cannot pop b immediately since there is no b-transition from the initial state of $\mathcal{B}$. Hence she violates the capacity constraint and loses the play.

However if the automaton $\mathcal{B}$ is complete, this would not happen. In any game $\mathcal{G}^\kappa(\mathcal{A}, \mathcal{B})$, where $\kappa \in \mathbb{N}^+$ and $\mathcal{B}$ is complete, whenever Spoiler forms a finite run over $w \in \Sigma^*$ that cannot be extended to any accepting run, Duplicator simply empties the buffer and forms any run over some word $v \sim w$. Duplicator then continues by simply reading the letter that is read by Spoiler in every round. She will win the play since Spoiler will not form an accepting run. Duplicator always forms an infinite run over a trace equivalent word.

Lemma 5.3.30. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBA over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ in which $\mathcal{B}$ is complete and $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ a function.* Duplicator *wins $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$ if f is Lipschitz continuous with constant c.*

Proof. Consider the following winning strategy for Duplicator in the Γ-game $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$. Suppose Spoiler and Duplicator have formed the runs $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$ respectively, and Spoiler extends his run one step to $r'_{\mathcal{A}} = r_{\mathcal{A}}ap$. If $r'_{\mathcal{A}}$ cannot be extended to some $\rho \in \mathsf{Dom}(f)$ then Duplicator extends her run maximally, i.e. to $r'_{\mathcal{B}}$ such that $\mathsf{word}(r'_{\mathcal{A}}) \sim \mathsf{word}(r'_{B})$. This is possible since $\mathcal{B}$ is complete. However in the case where $r'_{\mathcal{A}}$ can be extended to some $\rho \in \mathsf{Dom}(f)$ then we consider the same strategy as in the proof of Lemma 5.3.11: Duplicator skips her turn unless there exists $r'_{\mathcal{B}} := r_{\mathcal{B}}b_1q_1 \dots b_mq_m$ such that $f(\rho)$ is started with $r'_{\mathcal{B}}$. In such a case, Duplicator extends her run to such a maximal $r'_{\mathcal{B}}$.

We can show similarly as in the proof of Lemma 5.3.11 that Duplicator wins the game $\Gamma^\omega(\mathcal{A}, \mathcal{B}, f)$, in which the delay is unbounded. However, in this case we can also show that there is no round where the length difference of Spoiler's and Duplicator's run is more than c. We will show this by contradiction: suppose there is a round where Spoiler and Duplicator's run are respectively $r_{\mathcal{A}}$, $r_{\mathcal{B}}$, and $|r_{\mathcal{A}}| - |r_{\mathcal{B}}| > c$. By the construction of the winning strategy, this implies that we have two runs $\rho_1, \rho_2 \in \mathsf{Dom}(f)$ that can be extended from $r_{\mathcal{A}}$, but both $f(\rho_1), f(\rho_2)$ share the same prefix of no longer than $r_{\mathcal{B}}$: $d(f(\rho_1), f(\rho_2)) \geq 2^{-(|r_{\mathcal{B}}|+1)}$. Since $d(\rho_1, \rho_2) \leq 2^{-(|r_{\mathcal{A}}|+1)}$ and $|r_{\mathcal{A}}| - |r_{\mathcal{B}}| > c$, we have

$$d(f(\rho_1), f(\rho_2)) > 2^c \cdot d(\rho_1, \rho_2).$$

This contradicts that f is Lipschitz continuous with constant c. Hence there is no round where the length difference of Spoiler's and Duplicator's run is more than c. Duplicator also wins $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$. □

Conversely, if f is not Lipschitz continuous, Duplicator loses $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ for any $k \in \mathbb{N}$. Spoiler can play the accepting runs ρ_1, ρ_2 in $\mathcal{A}$ that share the same prefix of length n, but the output runs $f(\rho_1)$, $f(\rho_2)$ differ in their first $(n-k)$ transitions. In round n, Duplicator will form either the prefix of $f(\rho_1)$ or $f(\rho_2)$, and Spoiler can choose accordingly which run he should form. Duplicator will lose for not producing the image of Spoiler's run. We show this formally as follows.

Lemma 5.3.31. Duplicator *loses $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ for all $k \in \mathbb{N}$ if f is not Lipschitz continuous.*

Proof. By definition, if f is not Lipschitz continuous then for any $k \in \mathbb{N}$, there exist $\rho_1, \rho_2 \in \mathsf{AccRun}(\mathcal{A})$ such that $d(f(\rho_1), f(\rho_2)) > 2^k d(\rho_1, \rho_2)$. Hence let k be a number and ρ_1, ρ_2 be such accepting runs. Let $n \in \mathbb{N}$ such that $2^{-(n+1)} = d(\rho_1, \rho_2)$. The winning strategy for Spoiler in the game $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ is to first play ρ_1. At the end of round n, Duplicator forms a run r of length at least $n - k$. In the case where r is not a prefix of $f(\rho_1)$, Spoiler keeps playing ρ_1 for the rest of the play. Otherwise he continues by playing ρ_2. This is possible since ρ_1, ρ_2 share the same prefix of length n. In the first case, Spoiler wins because Duplicator does not form the f-image of ρ_1. In the second case, since $d(f(\rho_1), f(\rho_2)) > 2^{-(n-k-1)}$, $f(\rho_1), f(\rho_2)$ differ already at least in their first $(n-k)$ transitions. Since r is of length $n - k$ and r is a prefix of $f(\rho_1)$, r is not a prefix of $f(\rho_2)$. Spoiler wins the play. □

Now in the case where Duplicator wins the game $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ where $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ is trace-preserving, the completeness of $\mathcal{B}$ is still not sufficient to derive buffered simulation $\mathcal{A} \sqsubseteq^{k', \dots, k'} \mathcal{B}$ for some $k' \in \mathbb{N}$. Recall again the two automata $\mathcal{A}$,

$\mathcal{B}$ from Example 5.3.22 and a trace-preserving function $f(\rho) = q_0a(q_1b)^\omega$ for all $\rho \in \mathsf{AccRun}(\mathcal{A})$. The automaton $\mathcal{B}$ is complete. Duplicator wins the Γ-game $\Gamma^k(\mathcal{A}, \mathcal{B}, f)$ for $k = 0$. However Duplicator loses $\mathcal{G}^{k,\dots,k}(\mathcal{A}, \mathcal{B})$ for any $k \in \mathbb{N}$. This however will not happen if the commutative degree of $\mathcal{A}$ is finite.

If the commutative degree of $\mathcal{A}$ is finite and $\mathcal{B}$ is complete then whenever Duplicator wins the Γ-game $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$ for some $c \in \mathbb{N}$ and f is trace-preserving, she also wins the buffered simulation game $\mathcal{G}^{k,\dots,k}(\mathcal{A}, \mathcal{B})$ for some $k \in \mathbb{N}$. We can even specify $k = \mathsf{max}\{c, d\}$ where $d \in \mathbb{N}$ is the commutative degree of $\mathcal{A}$. Consider the winning strategy in the Γ-game as described in the proof of Lemma 5.3.30 and the translation as in Lemma 5.3.14. If the winning strategy in the Γ-game tells Duplicator to extend her run to $r'_\mathcal{B} = r_\mathcal{B} b_1 q_1 \dots b_m q_m$ then in the Γ-game, Duplicator extends $r_\mathcal{B}$ to $r_\mathcal{B} b_1 b_1 \dots b_{m'} q_{m'}$ the maximal prefix of $r'_\mathcal{B}$ that satisfies: $\pi_i(w)$ is a prefix of $\pi_i(v)$, for all $i \in \{1, \dots, n\}$. We have seen in Lemma 5.3.14 that with such a strategy, Duplicator wins the buffered simulation game $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A}, \mathcal{B})$. We will show that in the case where the commutative degree of $\mathcal{A}$ is some $d \in \mathbb{N}$, Duplicator in fact only stores at most $\mathsf{max}\{c, d\}$ many letters to each buffers.

The winning strategy intuitively is as follows. Duplicator first waits to have a preview of Spoiler's move c steps. She stores at most c many letters to each buffer. Since Duplicator wins $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$, after a lookahead of c steps, she knows how she should extend her run. Let us assume that she should read a letter a. If the letter a is already in the buffer then she can pop it and move her pebble along some a-transition. However if a is not yet in the buffer, she waits for more rounds until Spoiler reads a. Since $\mathsf{Deg}(\mathcal{A}) = d$, the letter a will be read by Spoiler at most in the next $|d - c|$ rounds. Hence to win the play Duplicator only needs to store at most $k = \mathsf{max}\{c, d\}$ letters. We formally show this as follows.

Theorem 5.3.32. *Let $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ be a trace-preserving function for two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$ in which $\mathsf{Deg}(\mathcal{A}) = d \in \mathbb{N}$ and $\mathcal{B}$ is complete. If* Duplicator *wins $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$ for some $c \in \mathbb{N}$ then* Duplicator *wins $\mathcal{G}^{k,\dots,k}(\mathcal{A}, \mathcal{B})$ where $k = \mathsf{max}\{c, d\}$.*

Proof. Consider the winning strategy as we have defined before. In the proof of Lemma 5.3.14, such a strategy is winning for Duplicator in the game $\mathcal{G}^{\omega,\dots,\omega}(\mathcal{A}, \mathcal{B})$. We will additionally show that there is no round where one of the buffers is filled with $k + 1$ many letters. We will show this by contradiction. Suppose there is such a round. Let $r_\mathcal{A}, r_\mathcal{B}$ be the runs that are formed by Spoiler and Duplicator in this round. Hence $|r_\mathcal{A}| - |r_\mathcal{B}| > k$. Note that Duplicator does not extend her run longer than $r_\mathcal{B}$ because either the winning strategy in $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$ tells her to extend only to $r_\mathcal{B}$, or it actually tells her to extend to some run $r'_\mathcal{B}$ longer than $r_\mathcal{B}$, but $r_\mathcal{B}$ is the maximal prefix of $r'_\mathcal{B}$ that satisfies

$$\pi_i(\mathsf{word}(r_\mathcal{B})) \text{ is a prefix of } \pi_i(\mathsf{word}(r_\mathcal{A})), \tag{5.4}$$

for all $i \in \{1, \dots, n\}$. In the first case, since $|r_\mathcal{A}| - |r_\mathcal{B}| > c$, it contradicts that the strategy is winning in $\Gamma^c(\mathcal{A}, \mathcal{B}, f)$. In the second case, suppose $r'_\mathcal{B}$ is extended from $r_\mathcal{B}$ by reading $u = a_1 \dots a_m$, i.e. $r'_\mathcal{B} = r_\mathcal{B}\, a_1 p_1 \dots a_m p_m$. Since (5.4) holds, $|\mathsf{word}(r_\mathcal{B})|_{a_1} \leq |\mathsf{word}(r_\mathcal{A})|_{a_1}$. However, since $r_\mathcal{B}$ is the maximal prefix of $r'_\mathcal{B}$ that satisfies (5.4), we have

$$|\mathsf{word}(r_\mathcal{B})|_{a_1} = |\mathsf{word}(r_\mathcal{A})|_{a_1} = \ell$$

for some $\ell \in \mathbb{N}$. Otherwise $r_\mathcal{B} a_1 p_1$ also satisfies (5.4) and contradicts the maximality of $r_\mathcal{B}$. Now let w and v be the words that are produced respectively by Spoiler and Duplicator in the game $\mathcal{G}^{\omega,\ldots,\omega}(\mathcal{A},\mathcal{B})$. Hence $\mathsf{word}(r_{\mathcal{A}})$ and $\mathsf{word}(r'_{\mathcal{B}})$ are prefixes of w and v, respectively. Since we assume that f is trace-preserving, $w \sim v$. Let $i \in \mathsf{Pos}(w)$ and $j \in \mathsf{Pos}(v)$ be the positions of the $(\ell+1)$-th a_1 in w and v, respectively. Hence $i > |r_{\mathcal{A}}|$ and $j = |r_{\mathcal{B}}|+1$. Since $|r_{\mathcal{A}}| - |r_{\mathcal{B}}| > k$ and $k \geq d$, we have $i - j > d$. However, since $(i, j) \in \mathsf{Corr}_{w,v}$ and w is a word over some finite path in $\mathcal{A}$, this contradicts $\mathsf{Deg}(\mathcal{A}) = d$. Hence there is no round where one of the buffers is filled with more than k letters. Duplicator wins the buffered simulation game $\mathcal{G}^{k,\ldots,k}(\mathcal{A},\mathcal{B})$. □

We can put this lemma together with Lemma 5.3.30 to show that the existence of a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ implies buffered simulation $\mathcal{A} \sqsubseteq^{k,\ldots,k} \mathcal{B}$ for some $k \in \mathbb{N}$, if the commutative degree of $\mathcal{A}$ is finite and the automaton $\mathcal{B}$ is complete.

Theorem 5.3.33. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBA in which the commutative degree of $\mathcal{A}$ is finite and $\mathcal{B}$ is complete. $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ for some $\kappa \in \mathbb{N}^*$ if there exists a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

Proof. Suppose we have a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$ with Lipschitz constant c. Since f is Lipschitz continuous and $\mathcal{B}$ is complete, by Lemma 5.3.30 Duplicator wins the game $\Gamma^c(\mathcal{A},\mathcal{B},f)$. Moreover let $d = \mathsf{Deg}(\mathcal{A})$. Since f is trace-preserving, by Lemma 5.3.32, Duplicator also wins the game $\mathcal{G}^{k,\ldots,k}(\mathcal{A},\mathcal{B})$ where $k = \max(c,d)$. □

Together with Lemma 5.3.20, this theorem allows us to further refine the characterisation in Theorem 5.3.16 into the following corollary.

Corollary 5.3.34. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBA in which the commutative degree of $\mathcal{A}$ is finite and $\mathcal{B}$ is complete. $\mathcal{A} \sqsubseteq^{\kappa} \mathcal{B}$ for some $\kappa \in \mathbb{N}^+$ iff there exists a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

This corollary shows a characterisation of buffered simulation with bounded buffers by considering the existence of a Lipschitz continuous trace-preserving function. The characterisation holds if Spoiler's automaton $\mathcal{A}$ has a finite commutative degree and Duplicator's automaton $\mathcal{B}$ is complete. Note that the condition for $\mathcal{A}$ is a semantic condition and the condition for $\mathcal{B}$ is a syntactic one. We can check whether the automaton $\mathcal{B}$ is complete by looking at its structure. However it is not obvious whether we can also check whether the commutative degree of $\mathcal{A}$ is finite by looking at its structure. Hence one may ask if there is a syntactic characterisation for such a condition. In the following subsection, we will show that such a syntactic characterisation indeed can be obtained by slightly lifting the condition for automata that have a regular trace closure.

5.3.3 Cyclic-Path-Connected Automata

Let us first consider a characterisation of an automaton that has a regular trace closure. Note that if the language $L(\mathcal{A})$ is regular then its trace closure $[L(\mathcal{A})]$ might not be regular. For example, consider the automaton $\mathcal{A}$ in Figure 5.1 over $\hat{\Sigma} = (\{a,c\},\{a,b\})$. The language $L(\mathcal{A}) = (ab)^*c^\omega$ is regular, but its trace closure $[L(\mathcal{A})] = \{vc^\omega \mid v \in \{a,b\}^*, |v|_a = |v|_b\}$ is not. However, there is sufficient syntactic condition for automata that have a regular trace closure [CL87]. It uses the notion of *connected words*.

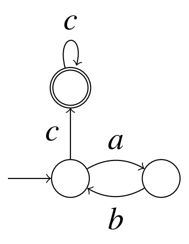


Figure 5.1: An automaton $\mathcal{A}$ with non-regular trace closure.

$a - b - c - d - e$ $\quad$ $a - b - c - d$ $\quad$ $a - b \quad d - e$

(a) G. (b) G_w. (c) $G_{w'}$.

Figure 5.2: Connected and non-connected dependency graphs.

Connected Words

Let w be a word over a distributed alphabet $\hat{\Sigma}$ and G the corresponding dependency graph of $\hat{\Sigma}$. The word w is called *connected* if the subgraph of G induced by the alphabet of w is connected.

Example 5.3.35. Consider a distributed alphabet $\hat{\Sigma} = (\{a, b\}, \{b, c\}, \{c, d\}, \{d, e\})$. Its corresponding dependency graph G then is given as in Figure 5.2a. Moreover, the word $w = adbc$ is connected, but the word $w = adbe$ is not. This is because the subgraph of G induced by $\Sigma_w = \{a, b, c, d\}$ as given in Figure 5.2b is connected and the one induced by $\Sigma_{w'} = \{a, b, d, e\}$ as given in 5.2c is not.

Intuitively, a word w is connected if there is no group of letters that are independent of all other letters in w. If a word w is not connected, we can partition the letters of w into two components such that each two letters from different components are independent of each other.

Proposition 5.3.36 ([DR95]). *Let $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ be a distributed alphabet and $G = (\Sigma, D)$ the corresponding dependency graph. If $w \in \Sigma^*$ is not connected then there exists a partition $\Sigma_w = \langle \Sigma', \Sigma'' \rangle$ such that*

$$w \sim \pi'(w)\pi''(w) \sim \pi''(w)\pi'(w), \tag{5.5}$$

where π', π'' are projections from Σ to Σ' and Σ'', respectively.

This proposition intuitively shows us that if w is not connected then we can partition w into two subsequences of letters w_1, w_2 such that the letters in w_1 and w_2 can swap their positions, i.e. $w \sim w_1 w_2 \sim w_2 w_1$.

The connectedness of a word is closely related to the commutative degree. If w is not connected then in the word w^k, the first letter of w^k can commute at least k steps to the right. Likewise for the last letter of w^k. It can commute at least k steps to the left. The reason is because by Proposition 5.3.36, we can partition the word w into two subsequences of letters w_1, w_2 of w in which the letters in w_1 and w_2 are independent of each other. Each letter of w either belongs to w_1 or w_2, but not both. Thus suppose the first letter of w belongs to w_1. Since $w^k \sim w_2^k w_1^k$, it is not hard to see that the first letter of w indeed commutes $|w_2^k|$ steps to the right.

Lemma 5.3.37. *If w is not connected then* $\mathsf{Deg}(w^k) \geq k$.

Proof. Suppose w is not connected. By Proposition 5.3.36, there exists a partition $\Sigma_w = \langle \Sigma', \Sigma'' \rangle$ such that (5.5) holds where π', π'' are projections from Σ to Σ' and Σ'', respectively. Moreover, we also have that

$$w^k \sim \pi'(w)^k \pi''(w)^k \sim \pi''(w)^k \pi'(w)^k.$$

In the case where the first letter of w belongs to Σ', i.e. $w(1) = a \in \Sigma'$, consider the word $v = \pi''(w)^k \pi'(w)^k$ that is trace equivalent to w. Let $n = |\pi''(w)^k|$ and $w' = w^k$. Hence $w'(1) = v(n+1) = a$ and $|w'(1)|_a = |v(1)\dots v(n+1)|_a = 1$ because $v(1), \dots, v(n) \neq a$. This implies $(1, n+1) \in \mathsf{Corr}_{w',v}$. Moreover, since $n \geq k$, we have $\mathsf{Deg}(w') \geq k$.

In the case where the first letter of w belongs to Σ'', we can show similarly that $\mathsf{Deg}(w^k) \geq k$ by considering the word $v = \pi'(w)^k \pi''(w)^k$. □

We can also extend this observation for the word uw in which u is connected and uw is not. By Proposition 5.3.36, we can partition the word uw into w_1, w_2 such that $uw \sim w_1 w_2 \sim w_2 w_1$. However, since u is connected, each letter of u either belongs to w_1 or w_2, but not both. Let us assume to w_1. There is at least one letter in w that belongs to w_2 since otherwise uw is connected. Now consider the word $u^k w$ that is obtained by repeating the word u, k many times before we read w. The letters in $u^k w$ that occur after position $|u^k|$ and belongs to w_2 can commute $|u^k|$ steps to the left. We formally show this as follows.

Lemma 5.3.38. *If w is connected and uw is not then*

- $\mathsf{Deg}(u^k w) \geq k$
- $\mathsf{Deg}(w u^k) \geq k$

Proof. We will show this for the first part. The second one can be shown similarly. Since uw is not connected, by Proposition 5.3.36, there is a partition $\Sigma_{uw} = \langle \Sigma', \Sigma'' \rangle$ such that $uw \sim \pi'(uw)\pi''(uw) \sim \pi''(uw)\pi'(uw)$ where π' and π'' are projections from Σ to Σ' and Σ'', respectively. This also implies

$$u^k w \sim u^{k-1} \pi'(uw) \pi''(uw) \sim u^{k-1} \pi''(uw) \pi'(uw).$$

Since u is connected, either $\Sigma_u \subseteq \Sigma'$ or $\Sigma_u \subseteq \Sigma''$, but not both.

Now suppose $\Sigma_u \subseteq \Sigma'$. We then have $\pi''(uw) = \pi''(w)$. This implies

$$u^k w \sim u^{k-1} \pi'(uw) \pi''(w) \sim \pi''(w)\, u^{k-1}\, \pi'(uw).$$

Let $v = \pi''(w)\pi'(u^k w)$ and $w' = u^k w$. Let i be the smallest position in w' such that $w'(i) \in \Sigma''$. Since $\Sigma_u \subseteq \Sigma'$, we have $i > |u^k|$. Let $a \in \Sigma''$ such that $w'(i) = a$. The smallest position j in v such that $v(j) \in \Sigma''$ is $j = 1$. Hence $w'(i) = v(1) = a$ and $|w'(1)\dots w'(i)|_a = |v(1)|_a = 1$. We have $(i, 1) \in \mathsf{Corr}_{w',v}$. Since $|u^k| \geq k$, we have $i > k$. Since $i - 1 \geq k$, we also have $\mathsf{Deg}(w') \geq k$.

In the case where $\Sigma_u \subseteq \Sigma''$, we can show similarly that $\mathsf{Deg}(u^k w) \geq k$ by considering the word $v = \pi'(w)\pi''(u^k w)$. □

Loop-Connected Automata

Now let us call an automaton *loop-connected* if every cycle is over a connected word. For example, consider the automaton $\mathcal{A}$ from Example 5.3.22. It is loop-connected since every cycle in $\mathcal{A}$ is over some word $w \in b^*$ and such a word w is connected with respect to any distributed alphabet. On the other hand, the automaton $\mathcal{A}$ in Figure 5.1 over $\hat{\Sigma} = (\{a, c\}, \{b, c\})$ is not loop-connected. There is a cycle from the initial state over ab and such a word is not connected with respect to the given $\hat{\Sigma}$. It is shown that loop-connectedness implies a regular trace closure.

Proposition 5.3.39 ([CL87])**.** *For any NBA $\mathcal{A}$ over $\hat{\Sigma}$, if $\mathcal{A}$ is loop-connected then the trace closure $[L(\mathcal{A})]$ is regular.*

Consider again the automaton $\mathcal{A}$ over $\hat{\Sigma}$ from Example 5.3.22. The automaton indeed has a regular trace closure. The trace closure of $\mathcal{A}$ is $[L(\mathcal{A})] = b^*ab^\omega$.

It turns out that loop-connectedness is a necessary condition for an automaton to have a finite commutative degree. If an automaton $\mathcal{A}$ is not loop-connected then for any $k \in \mathbb{N}$, we can consider a word that is produced by going through the non-connected cycle $k + 1$ many times. By Lemma 5.3.37, the commutative degree of such a word is more than k. Hence k is not the commutative degree of $\mathcal{A}$.

Lemma 5.3.40. *If the commutative degree of $\mathcal{A}$ is finite then $\mathcal{A}$ is loop-connected.*

Proof. We can show this by contraposition. Suppose $\mathcal{A}$ is not loop-connected. There is a cycle over a non-connected word w. For any $k \in \mathbb{N}$, consider the word w^{k+1} that is produced by a run that goes through such a cycle $k + 1$ many times. Since u is not connected, by Lemma 5.3.37 the commutative degree of w^{k+1} is greater than k. Hence for any $k \in \mathbb{N}$, $\mathsf{Deg}(\mathcal{A}) \geq k$. The commutative degree of $\mathcal{A}$ is infinite. □

The converse of this lemma however does not hold. If $\mathcal{A}$ is loop-connected, this does not necessarily imply that the commutative degree of $\mathcal{A}$ is finite. For example, consider the automaton $\mathcal{A}$ from Example 5.3.22. It is loop-connected, but the commutative degree of $\mathcal{A}$ is not finite as we have seen in Example 5.3.29.

Cyclic-Path-Connected Automata

To characterise automata with finite commutative degree, one reasonable attempt is to tighten the property of loop-connectedness. Instead of only for cycles, let us require every path in $\mathcal{A}$ that contains a cycle to be over a connected word and we will call such an automaton *cyclic-path-connected*.

Definition 5.3.41. An automaton $\mathcal{A}$ is cyclic-path-connected if for any path

$$p \xrightarrow{u} q \xrightarrow{v} q \xrightarrow{w} r$$

in which $u, w \in \Sigma^*$, and $v \in \Sigma^+$, the word uvw is connected.

For example, consider the automaton $\mathcal{A}$ from Example 5.3.22. The automaton $\mathcal{A}$ is loop-connected but not cyclic-path-connected since the path

$$p_0 \xrightarrow{a} p_1 \xrightarrow{b} p_1$$

contains a cycle but the word ab is not connected.

We can lift Lemma 5.3.40 for a cyclic-path-connected automaton. Intuitively, if an automaton $\mathcal{A}$ is loop-connected but not cyclic-path-connected then for any $k \in \mathbb{N}$, we simply consider the word that is produced by a non-connected cyclic path that goes over a certain cycle $k+1$ many times. Thus we either read $u^{k+1}w$ or wu^{k+1} in which uw is not connected, but u is connected. By Lemma 5.3.38, the commutative degree of such a word is more than k. Hence k is not the commutative degree of $\mathcal{A}$.

Theorem 5.3.42. *If the commutative degree of $\mathcal{A}$ is finite then $\mathcal{A}$ is cyclic-path-connected.*

Proof. Let us show this again by contraposition. Suppose $\mathcal{A}$ is not cyclic-path-connected. If $\mathcal{A}$ is loop-connected then by Lemma 5.3.40, $\mathcal{A}$ has an infinite commutative degree, and we have the desired result. Now suppose $\mathcal{A}$ is loop-connected. By definition, there exists a cyclic-path

$$p \xrightarrow{u} q \xrightarrow{v} q \xrightarrow{w} r$$

in which $u, w \in \Sigma^*$, and $v \in \Sigma^+$ in $\mathcal{A}$, but uvw is not connected. Since $\mathcal{A}$ is loop-connected, the word v is connected. Thus either uv or vw is not connected. In the first case, for every $k \in \mathbb{N}$, consider the word uv^{k+1} over the run

$$p \xrightarrow{u} q \xrightarrow{v} q \ldots \xrightarrow{v} q.$$

Since uv is connected and v is not, by the second part of Lemma 5.3.38, the commutative degree of such a word is at least $k+1$. Hence for any $k \in \mathbb{N}$, $\mathsf{Deg}(\mathcal{A}) > k$. We have $\mathsf{Deg}(\mathcal{A}) = \infty$. In the second case, we can also show similarly that $\mathsf{Deg}(\mathcal{A}) = \infty$ by considering the word $v^{k+1}w$ and the first part of Lemma 5.3.38. □

In contrast to Lemma 5.3.40, the converse of this theorem holds. However we need a more involved technique to show it. For the rest of this chapter, we will the converse of Theorem 5.3.42s. We will show that if an automaton $\mathcal{A}$ is cyclic-path-connected then whenever we visit a certain cycle over a word of length n, n many times, each two letters that are read before and after the repeated cycle do not commute with each other. Intuitively, the connected word produced by such a repeated cycle blocks other letters from commuting with each other. To show this formally we introduce the relation Block.

Relation Block

Intuitively, the relation Block is defined for a given word w over some distributed alphabet $\hat{\Sigma}$. It tells us the positions of letters of w that cannot commute with each other. Formally, let w be some finite word over $\hat{\Sigma}$, (Σ, D) the corresponding distributed alphabet, and

$$D_w = \{(i, j) \in \mathsf{Pos}(w)^2 \mid (w(i), w(j)) \in D,\ i \leq j\}.$$

The relation Block_w then is defined as the transitive closure of such a relation.

Example 5.3.43. Consider the word $w = cdbca$ in which the dependency between the letters is given by the dependency graph G from Figure 5.2a. We have

$$\begin{aligned}
D_w &= \{(1,2), (1,3), (1,4), (2,4), (3,4), (3,5)\} \\
D_w^2 &= D_w \cup \{(1,5)\} \\
D_w^* &= D_w^2
\end{aligned}$$

Hence $\mathsf{Block}_w = D_w^2$.

In this example, $(1, 5) \in \mathsf{Block}_w$ even though the letters a and c are independent of each other. This is because in between them, at position 3, we have b to which both a and c are dependent on. Hence the first and the fifth letters of w do not commute with each other.

In general, if $(i_0, i_n) \in \mathsf{Block}_w$, there are $n - 1 \geq 0$ many positions $i_1 < \ldots < i_{n-1}$ in between i_0 and i_n such that for all $k \in \{0, \ldots, n - 1\}$, the letters at positions i_k and i_{k+1} are dependent on each other. The letters at position i_0 and i_n cannot commute with each other.

Lemma 5.3.44. *If $(i, j) \in \mathsf{Block}_w$ then for all $v \sim w$, $\mathsf{Corr}_{w,v}(i) \leq \mathsf{Corr}_{w,v}(j)$.*

Proof. For the base case, let $(i, j) \in D_w$. Hence $(w(i), w(j)) \in D$. By Lemma 5.3.25, we have the desired result. For the inductive case, let $(i, j) \in D_w^n$ for some $n \in \mathbb{N}$. Hence there exists k, $i < k \leq j$ such that $(i, k) \in D_w^{n-1}$ and $(k, j) \in D_w$. Since $(i, k) \in D_w^{n-1}$, by induction hypothesis, for any $w' \sim w$,

$$\mathsf{Corr}_{w,w'}(i) \leq \mathsf{Corr}_{w,w'}(k). \tag{5.6}$$

Moreover since $(k, j) \in D_w$, we have $(w(k), w(j)) \in D$. Again by Lemma 5.3.25, for any $w'' \sim w$,

$$\mathsf{Corr}_{w,w''}(k) \leq \mathsf{Corr}_{w,w''}(j). \tag{5.7}$$

Now let v be some arbitrary word such that $v \sim w$. By (5.6), we have $\mathsf{Corr}_{w,v}(i) \leq \mathsf{Corr}_{w,v}(k)$. By (5.7), we have $\mathsf{Corr}_{w,v}(k) \leq \mathsf{Corr}_{w,v}(j)$. Hence $\mathsf{Corr}_{w,v}(i) \leq \mathsf{Corr}_{w,v}(j)$. □

The set Block_w is also complete in the sense that if a pair $i < j$ is not in Block_w then the letters at positions i and j can exchange their order. The reason is because by definition of Block_w, any letter in between i and j is independent of the ones at position i and j. Hence the letter at position i can commute $(j - i)$ many steps to the right.

Lemma 5.3.45. *Let $w \in \Sigma^*$ and $j \in \mathsf{Pos}(w)$.*

- *If there exist $i < j$ such that $(i, j) \notin \mathsf{Block}_w$ then there exists $v \sim w$ such that $(j, j-1) \in \mathsf{Corr}_{w,v}$.*
- *If there exist $i > j$ such that $(j, i) \notin \mathsf{Block}_w$ then there exists $v \sim w$ such that $(j, j+1) \in \mathsf{Corr}_{w,v}$.*

Proof. We will show this for the first part. The second part can be shown similarly. First, without loss of generality, let us assume that i is the biggest of such a position. Hence for all i' where $i < i' < j$,

$$(i', j) \in \mathsf{Block}_w. \tag{5.8}$$

This implies $(w(i), w(i')) \notin D$ for all i', $i < i' \leq j$ since otherwise $(i, j) \in \mathsf{Block}_w$ and contradicts our initial assumption. Thus we can apply Lemma 5.3.26 and obtain the desired result. □

We can lift this lemma to the case where we have several of such is. If there are positions $i_1, \ldots, i_m < j$ and all of them are not in the block relation with j then the position j can commute m steps to the left, likewise if there are $i_1, \ldots, i_m > j$ and all of them are not in the block relation with j. The position j then can commute m steps to the right.

Lemma 5.3.46. *Let $w \in \Sigma^*$ and $j \in \mathsf{Pos}(w)$.*

- *If there are m distinct positions $i_1, \ldots, i_m < j$ such that $(i_1, j), \ldots, (i_m, j) \notin \mathsf{Block}_w$ then there exists $v \sim w$ such that $(j, j - m) \in \mathsf{Corr}_{w,v}$.*
- *If there are m distinct positions $i_1, \ldots, i_m > j$ such that $(j, i_1), \ldots, (j, i_m) \notin \mathsf{Block}_w$ then there exists $v \sim w$ such that $(j, j + m) \in \mathsf{Corr}_{w,v}$.*

Proof. We will show this for the first part. The second one can be shown similarly. For the base case, let $m = 1$. By the first part of Lemma 5.3.45, we have the desired result. Now suppose $m > 1$. There exist m distinct positions $i_1, \ldots, i_m < j$ such that (i_1, j), …, (i_m, j) $\notin \mathsf{Block}_w$. Without loss of generality, let us assume that i_m is the largest of such positions. Hence for all i', $i_m < i' < j$, we have $(i', j) \in \mathsf{Block}_w$. This implies $(w(i_m), w(i')) \notin D$ for all i', $i_m < i' \leq j$, since otherwise $(i_m, j) \notin \mathsf{Block}_w$ and contradicts our assumption. By applying Lemma 5.3.26, we have a word $u \sim w$ where

$$\mathsf{Corr}_{w,u}(i) = \begin{cases} j & \text{if } i = i_m, \\ i - 1 & \text{if } i_m < i \leq j \\ i & \text{otherwise.} \end{cases} \tag{5.9}$$

Now in u, there are $(m - 1)$ many distinct positions $i_1, \ldots, i_{m-1} < j - 1$ and $(i_1, j - 1), \ldots, (i_{m-1}, j - 1) \notin \mathsf{Block}_u$. By induction hypothesis, there exists $v \sim u$ such that $(j-1, j-m) \in \mathsf{Corr}_{v,u}$. Since by (5.9), $(j, j-1) \in \mathsf{Corr}_{w,v}$, we have $(j, j-m) \in \mathsf{Corr}_{w,u}$. □

The block relation is indeed closely related to the commutative degree. By looking at how many positions i there are such that $i < j$ and $(i, j) \notin \mathsf{Block}_w$ we can determine how far the letter at position j can commute to the left. The other direction also holds similarly. By looking at how many positions i there are such that $i > j$ and $(j, i) \notin \mathsf{Block}_w$ we can determine the maximal length of how far the letter at position j can commute to the right. To show this formally, let us collect such positions i in the following sets.

$$\overline{\mathsf{Block}^-_w}(i) = \{j < i \mid (j, i) \notin \mathsf{Block}_w\} \tag{5.10}$$

$$\overline{\mathsf{Block}^+_w}(i) = \{j > i \mid (i, j) \notin \mathsf{Block}_w\} \tag{5.11}$$

Intuitively, these two sets collect the positions that are not in the Block relation with i. The positive and negative signs are simply used to indicate whether it occurs before or after the position i. In the following, we show that the size of the sets in (5.10) and (5.11) indeed are the commutative degree of the letter at position i.

Lemma 5.3.47. *Let w be some finite word over $\hat{\Sigma}$. We have*

$$\mathsf{Deg}^x_w(i) = |\overline{\mathsf{Block}^x_w(i)}|$$

for all $x \in \{+, -\}$ and $i \in \mathsf{Pos}(w)$.

Proof. We will show this for $b = -$. A similar argument can be used for $b = +$. Let $k_1 = \mathsf{Deg}^-_w(i)$ and $k_2 = |\overline{\mathsf{Block}^-_w(i)}|$.

If $k_1 < k_2$ then $k_1 < |\overline{\mathsf{Block}^-_w(i)}|$. There are at least $k_1 + 1$ many distinct positions in $\overline{\mathsf{Block}^-_w}(i)$. By definition of $\overline{\mathsf{Block}^-_w}(i)$, there are distinct positions $i_1, \ldots, i_{k_1+1} < i$ in w such that $(i_1, i), \ldots, (i_{k_1+1}, i) \notin \mathsf{Block}_w$. By Lemma 5.3.46, there exists w' such that $w' \sim w$ and $(i, i - (k_1 + 1)) \in \mathsf{Corr}_{w,w'}$. This implies $\mathsf{Deg}^-_w(i) > k_1$, which contradicts $\mathsf{Deg}^-_w(i) = k_1$.

However if $k_1 > k_2$ then $\mathsf{Deg}^-_w(i) > k_2$. By definition of $\mathsf{Deg}^-_w(i)$, there exist a word w' and a position $i' \in \mathsf{Pos}(w')$ such that $w \sim w'$, $(i,i') \in \mathsf{Corr}_{w,w'}$, and $i - i' > k_2$. Consider the set of positions

$$S = \{\ell < i \mid \mathsf{Corr}_{w,w'}(\ell) > i'\}$$

in w. If $|S| < i - i'$ then there are two distinct positions $\ell_1, \ell_2 < i$ where $\mathsf{Corr}_{w,v}(\ell_1) = \mathsf{Corr}_{w,v}(\ell_2)$. This is impossible by the definition of the correspondence relation since $w \sim v$. Hence $|S| \geq i - i'$ and thus $|S| > k_2$. Since there are only k_2 many positions in $\overline{\mathsf{Block}^-_w(i)}$ and more than k_2 position in S, there is at least one position that belongs to S but does not belong to $\overline{\mathsf{Block}^-_w(i)}$. Let j be such a position. Hence $(j,i) \in \mathsf{Block}_w$. However, since $j \in S$, $\mathsf{Corr}_{w,w'}(j) > \mathsf{Corr}_{w,w'}(i)$. This contradicts Lemma 5.3.44. Hence $k_1 = k_2$. □

By using this lemma, we can show that the commutative degree of $\mathcal{A}$ is finite whenever there exists a constant $d \in \mathbb{N}$ such that for every finite word w produced by $\mathcal{A}$ and position $i \in \mathsf{Pos}(w)$, the size of $\overline{\mathsf{Block}^-_w(i)}$ and $\overline{\mathsf{Block}^+_w(i)}$ are bounded by d. In the following, we will show such a constant d indeed exists if $\mathcal{A}$ is cyclic-path-connected.

Cyclic-Path-Connected ≈ Finite Commutative Degree

First we will show that if we have a cyclic-path-connected automaton and a word over some cyclic path i.e. a path that contains a cycle, each of the letters does not commute with at least one letter from the cycle. To illustrate this, consider the cyclic path

$$p \xrightarrow{w_1} q \xrightarrow{u} q \xrightarrow{w_2} p'$$

over $w = w_1uw_2$ in some cyclic-path-connected $\mathcal{A}$. Hence the word u is over a cycle. If there is a letter in w_1, u, or w_2 that commutes with all other letters in u then $\mathcal{A}$ is not cyclic-path-connected: if it is in u then u is not connected, if it is in $w_1 = a_1 \dots a_n$, suppose at position i, then $a_i \dots a_n u$ is not connected, and if it is in $w_2 = b_1 \dots b_m$, suppose at position j, then $ub_1 \dots b_j$ is not connected.

Lemma 5.3.48. *Let $p \xrightarrow{w_1} q \xrightarrow{u} q \xrightarrow{w_2} r$ be a path in a cyclic-path-connected automaton $\mathcal{A}$. Let $P_1, P_2, P_3 \in \mathsf{Pos}(w)$ where*

$$\begin{aligned}
P_1 &= \{1, \dots, |w_1|\},\\
P_2 &= \{|w_1| + 1, \dots, |w_1u|\},\\
P_3 &= \{|w_1u| + 1, \dots, |w_1uw_2|\}.
\end{aligned}$$

The following holds.

- *For all $i \in P_1$, there exists $j \in P_2$ such that $(i,j) \in \mathsf{Block}_w$.*
- *For all $j \in P_3$, there exists $i \in P_2$ such that $(i,j) \in \mathsf{Block}_w$.*

Proof. We will show the proof for the first part. The second one can be shown similarly. Let $i \in P_1$, $P'_1 = P_1 \setminus \{1, \dots, i\}$, $n = |w_1|$, $w = w_1uw_2$, and $w' = w(i)w(i+1)\dots w(n)$. Since $w'u$ is over a cyclic-path, $w'u$ is connected. There exists $j \in P'_1 \cup P_2$ such that $(w(i), w(j)) \in D$. If $j \in P_2$ then $(i,j) \in \mathsf{Block}_w$. If $j \in P'_1$ then by induction hypothesis, there exists $j' \in P_2$ such that $(j,j') \in \mathsf{Block}_w$. Since $(i,j) \in \mathsf{Block}_w$, we have $(i,j') \in \mathsf{Block}_w$. □

Let us call a path r in some automaton $\mathcal{A}$ a *wall* if for every path that is extended from r, the letters that are read before and after r do not commute with each other.

Definition 5.3.49. A path $r = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$ over $w = a_1 \dots a_n$ in $\mathcal{A}$ is called a *wall* if for every path r' extended from r, i.e.

$$r' = q \xrightarrow{w_1} p_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} p_n \xrightarrow{w_2} q'$$

over $v = w_1 w w_2$, we have $(i, j) \in \mathsf{Block}_v$ for all $i \in \{1, \dots, |w_1|\}$ and $j \in \{|w_1 w| + 1, \dots, |w_1 w w_2|\}$.

Example 5.3.50. Consider the following automaton over $\hat{\Sigma} = (\{a, b\}, \{b, c\}, \{c, d\})$.

$$\begin{tikzcd}
& & p_2 \arrow[d, bend left, "b"] & \\
\arrow[r] & p_0 \arrow[r, "a"'] & p_1 \arrow[u, bend left, "c"] \arrow[r, "d"'] & p_3
\end{tikzcd}$$

The corresponding dependency graph of $\hat{\Sigma}$ is $G : a - b - c - d$. If we consider the simple cycle $c = p_1 \xrightarrow{c} p_2 \xrightarrow{b} p_1$ then the path

$$p_0 \xrightarrow{a} p_1 \xrightarrow{c} p_2 \xrightarrow{b} p_1 \xrightarrow{d} p_3$$

is extended from c. It is over $w = acbd$. However since $(1, 4) \notin \mathsf{Block}_w$, the cycle c is not a wall.

On the other hand, the cycle

$$r = p_1 \xrightarrow{c} p_2 \xrightarrow{b} p_1 \xrightarrow{c} p_2 \xrightarrow{b} p_1$$

that goes through c twice, is a wall. For every path that is extended from r, every pair of letters that are read before and after r does not commute with each other. Consider the extended path r' from r which is over the word $v = acbcbd$. The first and the last letter of v do not commute with each other: we have $(1, 6) \in \mathsf{Block}_v$ since $(1, 3), (3, 4), (4, 6) \in D_v$.

Now let us denote with $C(\mathcal{A})$ the set of simple cycles in $\mathcal{A}$. By simple cycle, we mean a path that does not visit the same state twice except the first and the last states. Formally,

$$C(\mathcal{A}) = \{p_1 \xrightarrow{a_1} \dots \xrightarrow{a_k} p_{k+1} \mid p_1 \neq \dots \neq p_k \text{ and } p_1 = p_{k+1}\}.$$

For any automaton $\mathcal{A}$ with n states and m simple cycles, if we have a path of length $n^2 m$ then there exists a simple cycle that we have visited at least n many times. This is a simple consequence from the fact that for every path of length n, there exists a state that has been visited at least twice.

Proposition 5.3.51. *Let $\mathcal{A}$ be an automaton, $n = |\mathcal{A}|$, and $m = |C(\mathcal{A})|$. For every path r in $\mathcal{A}$ of length $n^2 m$, there exists a simple cycle $c \in C(\mathcal{A})$ that is visited by r at least n many times.*

Proof. Suppose r is a path of length $n^2 m$ in $\mathcal{A}$. Hence there exists a state $q \in Q^{\mathcal{A}}$ that is visited at least $nm + 1$ many times in r. In other words, the path r visits at least nm many cycles from q. Since every cycle is either simple or contains a simple cycle, the path r also visits at least nm many simple cycles. Since there are only m many different simple cycles in $\mathcal{A}$, there exists a simple cycle c that is visited at least n many times in r. □

Any path r of length $|\mathcal{A}|^2 \cdot |C(\mathcal{A})|$ in some cyclic-path-connected automaton $\mathcal{A}$ is a wall. The main reason is because such a path visits a simple cycle, suppose over u, at least $n = |\mathcal{A}|$ many times. By Lemma 5.3.48, each letter that is read before r does not commute with any letter from the first u and each letter that is read after r does not commute with any letter from the last u. The letters in the first and last us, suppose at positions i_1 and i_n, however, do not commute with each other. This is because u is connected. There are letters at positions $i_2, \ldots, i_{n-1}$, that occur in the second u, up to $(n-1)$-th u, such that the ones at positions i_j and i_{j+1} are dependent on each other for all $j \in \{1, \ldots, n-1\}$. We formally show this as follows.

Lemma 5.3.52. *Let $\mathcal{A}$ be an NBA over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. If $\mathcal{A}$ is cyclic-path-connected then every path of length $|\mathcal{A}|^2 \cdot |C(\mathcal{A})|$ is a wall.*

Proof. Let r be a path of length $|\mathcal{A}|^2 \cdot |C(\mathcal{A})|$ and $n = |\mathcal{A}|$. By Proposition 5.3.51, the path r can be seen as

$$p \xrightarrow{v_0} q \xrightarrow{u} q \xrightarrow{v_1} q \xrightarrow{u} q \ldots \xrightarrow{v_n} p'$$

where u is a word over a simple cycle from q. The word u occurs at least n many times in r. If we consider a path r' that is extended from r then r' is over $w = w_1(v_0uv_1u\ldots v_n)w_2$ for some $w_1, w_2 \in \Sigma^*$. Let i_0 and j_0 be two positions that occur in w_1 and w_2 respectively, i.e. $i_0 \in \{1, \ldots, |w_1|\}$ and $j_0 \in \{|w_1(v_0uv_1\ldots v_n)| + 1, \ldots, |w|\}$. For simplicity, let us call the word u that occurs in between v_0 and v_1 the first u, the one in between v_1 and v_2 the second u, and so on. Moreover for any $k \in \{1, \ldots, n\}$, let m_k be the starting position of the k-th u, i.e. $m_k = |w_1v_0u\ldots uv_{k-1}|$ and let P_k be the set of positions of the k-th u, i.e. $P_k = \{m_k + 1, \ldots, m_k + |u|\}$. By the first part of Lemma 5.3.48, since we can see the path r' as

$$p_1 \xrightarrow{w_1\, v_0} q \xrightarrow{u} q \xrightarrow{v_1u\ldots v_n\, w_2} p_2,$$

every letter in w_1 does not commute with at least one letter in the first u: there is $i_1 \in P_1$ such that

$$(i_0, i_1) \in \mathsf{Block}_w. \tag{5.12}$$

By the second part of Lemma 5.3.48, since we can also see the path r' as

$$p_1 \xrightarrow{w_1\, v_0u\ldots v_{n-1}} q \xrightarrow{u} q \xrightarrow{v_n\, w_2} p_2,$$

every letter in w_2 does not commute with at least one letter in the n-th u: there is $j_1 \in P_n$ such that

$$(j_1, j_0) \in \mathsf{Block}_w. \tag{5.13}$$

We will show that the letters in the first and the n-th u do not commute with each other. First note that there is a path from $w(i_1)$ to $w(j_1)$ in the dependency graph G_u since $w(i_1)$, $w(j_1) \in \Sigma_u$ and the word u is connected. Moreover, there exists such a path of length less than n since $|G_u| \leq n$. Note that $|G_u| \leq n$ because u is a word over a simple cycle. Hence the length, as well as, the number of different letters of u are less than n. Let us choose $i_2 \in P_2, i_3 \in P_3, \ldots, i_{n'} \in P_{n'}$, $n' \leq n$, i.e. the positions in the second u, the third u, etc., such that

$$w(i_1) \to w(i_2) \to w(i_3) \to \ldots \to w(i_{n'}) \to w(j_1)$$

is the shortest path from $w(i_1)$ to $w(j_1)$ in the dependency graph $G_u = (\Sigma_u, E_u)$. Since u is connected, for all $a, b \in \Sigma_u$, $(a, b) \in E_u$ implies $(a, b) \in D$. Hence

$$(w(i_1), w(i_2)), \ \ldots, \ (w(i_{n'}), w(j_1)) \in D.$$

Since $i_1 < i_2 < \ldots < i_{n'} < j_1$, we have $(i_1, i_2), (i_2, i_3), \ldots, (i_{n'}, j_1) \in \mathsf{Block}_w$. This implies $(i_1, j_1) \in \mathsf{Block}_w$. By (5.12) and (5.13), $(i_0, j_0) \in \mathsf{Block}_w$. Thus r is a wall. □

Hence if we have a cyclic-path-connected automaton $\mathcal{A}$ and a word w produced by $\mathcal{A}$, all letters in w do not commute with any letter that occurs in more than $|\mathcal{A}|^2 \cdot |C(\mathcal{A})|$ positions away. In other words, each letter in w can only commute at most $|\mathcal{A}|^2 \cdot |C(\mathcal{A})|$ positions to the left or right.

Lemma 5.3.53. *Let $\mathcal{A}$ be a cyclic-path-connected automaton over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, $p_1 a_1 p_2 \ldots a_k p_{k+1}$ is a finite run on $\mathcal{A}$ over $w = a_1 \ldots a_k$, and $i \in \mathsf{Pos}(w)$. We have*

- $|\overline{\mathsf{Block}^+_w(i)}| \leq |\mathcal{A}|^2 \cdot |C(\mathcal{A})|$
- $|\overline{\mathsf{Block}^-_w(i)}| \leq |\mathcal{A}|^2 \cdot |C(\mathcal{A})|$

Proof. Let $n = |\mathcal{A}|$ and $m = |C(\mathcal{A})|$. We will show this for the first part. The second part can be shown similarly.

First note that by definition, $\overline{\mathsf{Block}^+_w(i)} \subseteq \{i+1, \ldots, k\}$. Hence if $k < i + n^2 m + 1$, there are at most $k - i$ many positions in such a set. We have $|\overline{\mathsf{Block}^+_w(i)}| \leq n^2 \cdot m$.

However, if $k \geq i + n^2 m + 1$, by Lemma 5.3.52, the paths $p_{i+1} a_{i+1} \ldots p_{i+n^2 m+1}$ are walls in $\mathcal{A}$. Hence $(i, j) \in \mathsf{Block}_w$ for all $j > i + n^2 m$. This implies $\overline{\mathsf{Block}^+_w(i)} \subseteq \{i+1, \ldots, i+n^2 m\}$. Thus, we also have $|\overline{\mathsf{Block}^+_w(i)}| \leq n^2 \cdot m$. □

Together with Lemma 5.3.47 this lemma shows that for any cyclic-path-connected automaton $\mathcal{A}$, the commutative degree of $\mathcal{A}$ is finite. Hence the reverse direction of Theorem 5.3.42 also holds. Cyclic-path-connectedness syntactically characterises the automata with finite commutative degree.

Corollary 5.3.54. *$\mathcal{A}$ is cyclic-path-connected iff the commutative degree of $\mathcal{A}$ is finite.*

Proof. The right-to-left direction holds by Theorem 5.3.42. For the other direction, suppose $\mathcal{A}$ is cyclic-path-connected. Let $d = |\mathcal{A}|^2 \cdot |C(\mathcal{A})|$. Moreover let $r = p_1 a_1 \ldots a_k p_{k+1}$ be some arbitrary finite run in $\mathcal{A}$ over $w = a_1 \ldots a_k$ and $i \in \mathsf{Pos}(w)$. By Lemma 5.3.53,

$$|\overline{\mathsf{Block}^b_w(i)}| \leq d$$

for all $b \in \{+, -\}$. By Lemma 5.3.47, we have $\mathsf{Deg}^+_w(i)$, $\mathsf{Deg}^-_w(i) \leq d$. Since we consider a word w over some arbitrary run in $\mathcal{A}$ and position i of w, this holds for all word w produced by $\mathcal{A}$ and position i of w. Hence by definition, $\mathsf{Deg}(\mathcal{A}) \leq d$. □

By using this corollary, the characterisation of buffered simulation with bounded buffers in Theorem 5.3.33 can be revised as follows.

Corollary 5.3.55. *For any two NBA $\mathcal{A}$, $\mathcal{B}$ in which $\mathcal{A}$ is cyclic-path-connected and $\mathcal{B}$ is complete, $\mathcal{A} \sqsubseteq^{k,\ldots,k} \mathcal{B}$ for some $k \in \mathbb{N}$ iff there exists a Lipschitz continuous trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$.*

$$\begin{tikzcd}
L(\mathcal{A}) \subseteq [L(\mathcal{B})] \arrow[r, Leftrightarrow, "\text{Lem. 5.3.3}"] & \exists \text{ trace-preserving (TP) } f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B}) \\
\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B} \arrow[u, Rightarrow, "\text{Thm. 5.2.8}"] \arrow[r, Leftrightarrow, "\text{Thm. 5.3.16}"] & \exists \text{ continuous TP } f \colon \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B}) \\
\mathcal{A} \sqsubseteq^{k,\dots,k} \mathcal{B} \arrow[u, Rightarrow] \arrow[r, Rightarrow, "\text{Lem. 5.3.20}"] & \exists \text{ Lipschitz continuous TP } f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B}) \\
 & + \\
 & \mathcal{A} \text{ is cyclic-path-connected and } \mathcal{B} \text{ is complete} \arrow[uul, Rightarrow, "\text{Cor. 5.3.55}"]
\end{tikzcd}$$

Figure 5.3: Topological characterisation of buffered simulation.

5.4 Summary

We have shown that for any two NBA $\mathcal{A}$, $\mathcal{B}$ over Σ, buffered simulation with one bounded buffer can be used to approximate language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$ incrementally. We check whether buffered simulation $\mathcal{A} \sqsubseteq^k \mathcal{B}$ holds by starting from $k = 0$ and gradually increasing the parameter k. Language inclusion then holds if $\mathcal{A} \sqsubseteq^k \mathcal{B}$ holds for some $k \in \mathbb{N}$.

A similar approach also holds for approximating trace closure inclusion $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$. Given two NBA $\mathcal{A}$, $\mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \dots, \Sigma_n)$, we check whether buffered simulation $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ holds by starting from $\kappa = (0, \dots, 0)$ and gradually increasing the capacity vector κ. Trace closure inclusion then holds if $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ holds for some $\kappa \in \mathbb{N}^n$.

The characterisation of buffered simulation is summarised in Figure 5.3. First we have shown that the inclusion $L(\mathcal{A}) \subseteq [L(\mathcal{B})]$, and equivalently $[L(\mathcal{A})] \subseteq [L(\mathcal{B})]$, can be characterised by the existence of a trace-preserving function $f : \mathsf{AccRun}(\mathcal{A}) \to \mathsf{AccRun}(\mathcal{B})$. We can lift this characterisation for buffered simulation $\mathcal{A} \sqsubseteq^{\omega,\dots,\omega} \mathcal{B}$ by considering the existence of such a trace-preserving function f that is also continuous. Furthermore, in the case where $\mathcal{A}$ is cyclic-path-connected and $\mathcal{B}$ is complete, we can lift the characterisation for buffered simulation with bounded buffers, i.e. $\mathcal{A} \sqsubseteq^{k,\dots,k} \mathcal{B}$ for some $k \in \mathbb{N}$, by considering the existence of such a trace-preserving function f that is not only continuous but also Lipschitz continuous.

Chapter 6

Conclusion

We have studied buffered simulation, an extension of the standard fair simulation between two Büchi automata where DUPLICATOR is allowed to store the letters that are read by SPOILER to the buffers before she executes them in her structure. The possibility to use buffers allows DUPLICATOR to have a preview of SPOILER's moves. She can mimic SPOILER's run more accurately than in the standard fair simulation.

Buffered simulation with one buffer approximates language inclusion better than standard simulation. We can even use it to incrementally approximate language inclusion. In the case where multiple buffers are involved, buffered simulation approximates trace closure inclusion, a more general problem than language inclusion which models the verification problem of concurrent systems. We can also use buffered simulation with multiple buffers to incrementally approximate trace closure inclusion.

The computational complexity of solving buffered simulation varies depending on the number of the buffers and their sizes. It ranges from PTIME, in the case where all buffers are bounded, to somewhere between $\mathbb{B}\Sigma_1^1$ and Δ_2^1 in the analytical hierarchy, in the case where some buffers are unbounded. There are also some special cases of buffered simulation in which the complexities are complete for the classes EXPTIME and PSPACE. This wide span of complexity classes can be seen as an indication of the richness of the framework.

We justify the framework of buffered simulation theoretically by showing a characterisation with the notion of continuity from topology. Buffered simulation in general can be characterised by the existence of a continuous function that witnesses trace closure or language inclusion. This charaterisation then allows us to classify pairs of automata in which their language or trace closure inclusion can be shown with buffered simulation. The characterisation can even be refined for the case where all buffers are bounded by considering the existence of such a function that is also Lipschitz continuous. The refined characterisation, however, only holds for some restricted class of automata namely the cyclic-path-connected automata.

In the following, we list some open problems regarding buffered simulation.

Buffered Simulation on Other ω-Regular Automata. In this work, we only have considered buffered simulation between two Büchi automata. One possible further work is to consider buffered simulation on other ω-regular automata such as generalised Büchi, parity, Rabin, Street, or Muller automata. One may get tempted to solve such a simulation by first translating the automata to Büchi automata by using a standard translation such as the one in [GTW02]. For example, consider a parity automaton $\mathcal{A}$ with priorities

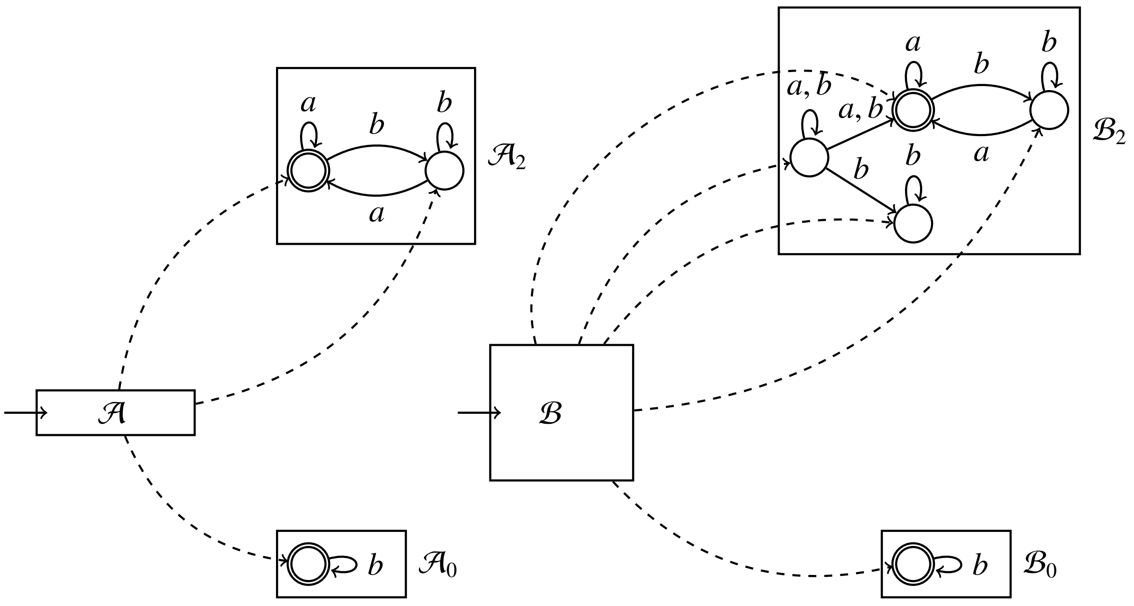
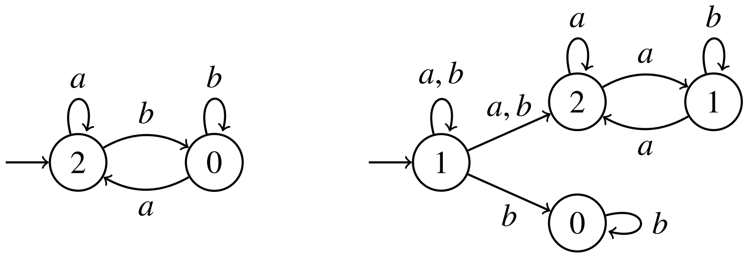


Figure 6.1: Translation of parity automata $\mathcal{A}$, $\mathcal{B}$ to Büchi automata.

$\{0, 1, \ldots, n\}$ and the standard translation in [GTW02]. We then obtain a Büchi automaton $\mathcal{A}'$ from $\mathcal{A}$ by adding m copies of $\mathcal{A}$: $\mathcal{A}_0, \mathcal{A}_2, \ldots \mathcal{A}_m$ where from the original part of $\mathcal{A}$, we can choose non-deterministically to continue to either $\mathcal{A}_0, \mathcal{A}_2, \ldots$, or $\mathcal{A}_m$. Each automaton $\mathcal{A}_i$ consists of states with priority at most i. Moreover in $\mathcal{A}_i$, the states with priority i are considered to be accepting. Now consider the simulation problem on these translated automata. It turns out that such a translation is not faithful, even with respect to fair simulation. To exemplify this, consider the pair of parity automata $\mathcal{A}$ (left) and $\mathcal{B}$ (right) as follows.



Both of the automata are universal. They accept all infinite words over a and b. Consider their translations to Büchi automata $\mathcal{A}'$, $\mathcal{B}'$ that are given in Figure 6.1. In this case, if we consider the fair simulation game between $\mathcal{A}'$, $\mathcal{B}'$ then DUPLICATOR wins. She simply stays and moves accordingly in $\mathcal{B}$ if SPOILER stays in $\mathcal{A}$. She moves to $\mathcal{B}_0$ or $\mathcal{B}_2$ only if SPOILER moves to $\mathcal{A}_0$ or $\mathcal{A}_2$, respectively. DUPLICATOR, however, loses the standard fair simulation game in the original automata $\mathcal{A}$, $\mathcal{B}$. This is because at one point she has to reveal the even priority that she will see infinitely often. If DUPLICATOR chooses 2 then SPOILER can continue by reading b^ω and if DUPLICATOR chooses 0 then SPOILER can continue by reading a^ω. Hence DUPLICATOR loses for not producing an accepting run. This shows that the translation, even though it preserves the language, does not preserve non-simulation.

Hence we first need to find a faithful translation to Büchi automata that also preserves simulation. Alternatively, we can also try to solve simulation between various ω-regular automata directly without translating them first to Büchi automata.

Practical Implementation of Buffered Simulation. In Chapter 5, we have seen that bounded buffered simulation can be used to approximate language and trace closure inclusion incrementally. The incremental algorithm basically starts with some small vector capacity $\kappa \in \mathbb{N}^+$ and then checks buffered simulation successively by increasing κ. For each buffered simulation with a fixed capacity $\kappa \in \mathbb{N}^+$, we can reduce it to a parity game with three priorities.

It is then interesting to see how this works in practice. One possible further work is to implement the incremental algorithm by using some parity game solver as a backend. For example, we can use PGSolver [FL09] or Oink [vD18] that already implement various algorithms for solving parity games. We then compare the implementation of buffered simulation with other approximation methods such as the incremental algorithm induced by the multi-pebble simulation or the flushing variants. We might also compare them with some complete approximation methods such as the ones that are based on complementation [SVW87], rank-based method [FKWV13, Sch09], or Ramsey based method [ACC+11, ACC+10, FV10].

An Upper Bound for Incremental Approximation. Recall that the incremental approximation for language and trace closure inclusion given in Chapter 5 might not terminate. If we consider two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ where for any $\kappa \in \mathbb{N}^n$, $\mathcal{A} \not\sqsubseteq^\kappa \mathcal{B}$, then the algorithm runs forever. Unlike the incremental approximation induced by multi-pebble simulation, we do not have an upper bound $\kappa_0 \in \mathbb{N}^n$ that tells us to stop increasing the capacity vector κ.

It is then interesting to know whether such a bound exists, i.e. whether there is $\kappa_0 \in \mathbb{N}^n$ such that if $\mathcal{A} \not\sqsubseteq^{\kappa_0} \mathcal{B}$ then $\mathcal{A} \not\sqsubseteq^\kappa \mathcal{B}$ for all $\kappa > \kappa_0$. Having such an upper bound κ_0 will give us a terminating incremental approximation since increasing κ further will not make the approximation for language or trace closure inclusion any better. We can stop the algorithm when κ reaches κ_0.

Undecidability of $\sqsubseteq^\kappa_{\textsf{Flush}}$. In this work, we have shown that deciding buffered simulation for the general case is highly undecidable. Given two NBA $\mathcal{A}, \mathcal{B}$ over $\hat{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ and a vector capacity $\kappa \in (\mathbb{N} \cup \{\omega\})^n$, the problem of deciding $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ is highly undecidable. One possible further work is to consider its flushing variant. Intuitively, solving the flushing variant of buffered simulation should be easier than the general case since Duplicator's moves are more restricted. She is required to empty the entire buffer every time she decides to move. In the case of one buffer, the flushing variant has a slightly better complexity. Deciding $\mathcal{A} \sqsubseteq^\omega \mathcal{B}$ and $\mathcal{A} \sqsubseteq^\omega_{\textsf{Flush}} \mathcal{B}$ are respectively EXPTIME and PSPACE-complete.

It is reasonable to ask whether this also holds in the general case, whether deciding the flushing variant $\mathcal{A} \sqsubseteq^\kappa_{\textsf{Flush}} \mathcal{B}$ is easier than deciding $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$. In Section 4.4, we have shown that deciding $\mathcal{A} \sqsubseteq^\kappa \mathcal{B}$ is highly undecidable by a reduction from the recurrent octant tiling problem. However, we cannot use the same technique for the flushing variant since the reduction heavily relies on Duplicator's ability to constantly keep some content in the buffer, namely the tiling of the last row. Hence it would also be interesting to see whether deciding the flushing variant $\mathcal{A} \sqsubseteq^\kappa_{\textsf{Flush}} \mathcal{B}$ is still decidable or "less" undecidable than the general case.

Buffered Simulation on Non-ω-Regular Automata. Besides considering buffered simulation between automata that recognise regular languages, we can also consider buffered simulation between two automata that recognise non-regular languages such as pushdown automata or visibly pushdown automata. Pushdown automata are automata with a stack [HU67, HMU06]. They recognise the class of context-free languages. Visibly pushdown automata on the other hand are more restricted [AM04, AM09]. They are pushdown automata in which we cannot push and pop the stack with the same input letter. They recognise the class of visibly context-free languages which is strictly included in the context-free languages. Deciding language inclusion between pushdown automata over finite words is known to be undecidable and the one between visibly pushdown automata is EXPTIME-complete [Löd14, AM04].

Similarly to the case of regular languages, we can also consider pushdown and visibly pushdown automata for infinite words. The complexity result of deciding language inclusion in the finite case can be lifted to the infinite case. Deciding language inclusion between pushdown automata over infinite words is undecidable and the one between visibly pushdown automata is EXPTIME-complete [AM04]. Moreover, solving the standard fair simulation between pushdown automata as well as visibly pushdown automata over infinite words is also EXPTIME-complete [KM02, Srb06].

Now if we consider buffered simulation with a single bounded buffer between two visibly pushdown automata over infinite words, it is not hard to see that the problem is also decidable in EXPTIME. We can reduce it to a parity game on a pushdown system. We model the stacks that are produced by Spoiler and Duplicator, whose heights differ by at most $k + 1$, as one stack. The EXPTIME-hardness then follows from the standard fair simulation between visibly pushdown automata.

Nevertheless, if we consider an unbounded buffer, it is not obvious anymore whether buffered simulation between visibly pushdown automata is still decidable. The height difference between Spoiler's and Duplicator's stacks is unbounded. Note that in the case of pushdown automata, buffered simulation with an unbounded buffer is undecidable. This is because simulation with an unbounded buffer captures language inclusion for finite words. Duplicator simply can wait until Spoiler produces a word in its full length before she moves. Since language inclusion between two pushdown automata for finite words is undecidable, buffered simulation with an unbounded buffer between pushdown automata for infinite words is also undecidable. However this might not be the case for visibly pushdown automata. Deciding language inclusion between two visibly pushdown automata for finite words is still decidable. Thus deciding simulation with an unbounded buffer between two visibly pushdown automata over infinite words might be still decidable. We first can ask whether the technique using the lasso game as in Section 4.3 can be extended to the case of visibly pushdown automata.

Application of Cyclic-Path-Connected Property. In Section 5.3, we have seen a restricted class of automata, namely cyclic-path-connected automata. If an automaton is cyclic-path-connected then there exists a bound $k \in \mathbb{N}$ such that for every word over a finite run, each of its letters does not commute more than k steps. This property might have an application in some other area than buffered simulation. For example, it might be useful in the area of Message Sequence Graphs (MSG) that is used to model the specification of communication between a system and its environment by means of message interchange [GR93, DH99]. In [MP99], it is shown that by restricting an MSG to be loop-connected, two undecidable problems concerning the correctness and consistency become decidable, namely in EXPSPACE. The reason is because a loop-connected MSG only allows local synchronisation instead of global synchronisation. Now if we consider a more restricted property such as cyclic-path-connected then a cyclic-path-connected MSG intuitively only allows bounded local synchronisation. Thus one reasonable question is to ask whether by restricting the MSG to be cyclic-path-connected, this would give us a better complexity than EXPSPACE for such problems concerning correctness and consistency.

Index

Bibliography

[ABH$^+$08] P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Computing simulations over tree automata. In C. R. Ramakrishnan and J. Rehof, editors, *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 93–108. Springer, 2008.

[ACC$^+$10] P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-H. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In T. Touili, B. Cook, and P. Jackson, editors, *Proceedings of the 22nd International Conference on Computer Aided Verification*, CAV'10, pages 132–147. Springer, 2010.

[ACC$^+$11] P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-H. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In J.-P. Katoen and B. König, editors, *Proceedings of the 22nd International Conference on Concurrency Theory*, CONCUR'11, pages 187–202. Springer, 2011.

[AM04] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceeding of the 36th Symposium on Theory of Computing*, STOC'04, pages 202–211. ACM, 2004.

[AM09] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):16:1–16:43, 2009.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[BF84] C. Burks and D. Farmer. Towards modeling DNA sequences as automata. *Physica D: Nonlinear Phenomena*, 10(1):157 – 167, 1984.

[BG03] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Transactions on Computational Logic*, 4(2):181–206, 2003.

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* (Perspectives in Mathematical Logic). Springer, 1997.

[Bir93] J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical systems theory*, 26(3):237–269, 1993.

[Boa97] P. V. E. Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, pages 331–363. Marcel Dekker Inc, 1997.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Proceedings of the 1st International Congress on Logic, Methodology and Philosophy of Science*, LMPS'60, pages 1–11. Stanford University Press, 1962.

[CF05] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005.

[Chl86] B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.

[CL87] M. Clerbout and M. Latteux. Semi-commutations. *Information and Computation*, 73(1):59–74, 1987.

[CM13] L. Clemente and R. Mayr. Advanced automata minimization. In *Proceeding of the 40th Symposium on Principles of Programming Languages*, POPL'13, pages 63–74. ACM, 2013.

[DH99] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. In *Proceeding of the 3rd IFIF International Conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS'99. Kluwer, 1999.

[DHWT92] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *Proceeding of the 3rd International Workshop on Computer Aided Verification*, CAV'91, pages 255–265. Springer, 1992.

[DR95] V. Diekert and G. Rozenberg. *The Book of Traces.* World Scientific Publishing Co., Inc., 1995.

[DR09] L. Doyen and J.-F. Raskin. Antichains for the automata-based approach to model-checking. *Logical Methods in Computer Science*, 5(1), 2009.

[EH00] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *Proceeding of the 11th International Conference on Concurrency Theory*, Concur'00, pages 153–167. Springer, 2000.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, SFCS '91, pages 368–377. IEEE Computer Society, 1991.

[Ete02] K. Etessami. A hierarchy of polynomial-time computable simulations for automata. In *Proceeding of the 13th International Conference on Concurrency Theory*, CONCUR'02, pages 131–144. Springer, 2002.

[EWS01] K. Etessami, T. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, ICALP '01, pages 694–707. Springer-Verlag, 2001.

[Fin12] O. Finkel. Three applications to rational relations of the high undecidability of the infinite post correspondence problem in a regular ω-language. *International Journal of Foundations of Computer Science*, 23(7):1481–1498, 2012.

[FKWV13] S. Fogarty, O. Kupferman, T. Wilke, and M. Y. Vardi. Unifying Büchi complementation constructions. *Logical Methods in Computer Science*, 9(1):248–263, 2013.

[FL09] O. Friedmann and M. Lange. Solving parity games in practice. In *Proceeding of the 7th International Symposium on Automated Technology for Verification and Analysis*, ATVA '09, pages 182–196. Springer-Verlag, 2009.

[FV09] S. Fogarty and M. Y. Vardi. Büchi complementation and size-change termination. In *Proceeding of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '09, pages 16–30. Springer-Verlag, 2009.

[FV10] S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *Proceeding of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'10, pages 205–220. Springer-Verlag, 2010.

[FW05] C. Fritz and T. Wilke. Simulation relations for alternating Büchi automata. *Theoretical Computer Science*, 338(1):275 – 314, 2005.

[GBS02] S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Proceeding of the 14th International Conference on Computer-Aided Verification*, CAV'02, pages 610–624. Springer, 2002.

[GR93] J. Grabowski and E. Rudolph. Message Sequence Chart (MSC) - A Survey of the new CCITT Language for the Description of Traces within Communication Systems. *CCITT*, pages 30–48, 1993.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., 2002.

[Har85] David Harel. Recurring dominoes: Making the highly undecidable highly understandable. In *Selected Papers of the International Conference on "Foundations of Computation Theory" on Topics in the Theory of Computation*, pages 51–71. Elsevier North-Holland, Inc., 1985.

[HHK$^+$16a] M. Hutagalung, N. Hundeshagen, D. Kuske, M. Lange, and É. Lozes. Multi-buffer simulations for trace language inclusion. In *Proceedings of the 7th International Symposium on. Games, Automata, Logics and Formal Verification*, GandALF'16, pages 213–227, 2016.

[HHK$^+$16b] M. Hutagalung, N. Hundeshagen, D. Kuske, M. Lange, and É. Lozes. Two-buffer simulation games. In *Proceedings Cassting Workshop on Games for the Synthesis of Complex Systems and 3rd International Workshop on Synthesis of Complex Parameters*, Cassting/SynCoP'16, pages 27–38, 2016.

[HHK$^+$18] M. Hutagalung, N. Hundeshagen, D. Kuske, M. Lange, and É. Lozes. Multi-buffer simulations: Decidability and complexity. *Inf. Comput.*, 262(2):280–310, 2018.

[HKR02] T. A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002.

[HKT10] M. Holtmann, L. Kaiser, and W. Thomas. Degrees of lookahead in regular infinite games. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'10, pages 252–266. Springer, 2010.

[HL11] Martin Hofmann and Martin Lange. *Automatentheorie und Logik*. eXamen.press. Springer, 2011.

[HLL13] M. Hutagalung, M. Lange, and E. Lozes. Revealing vs. concealing: More simulation games for Büchi inclusion. In *Proceeding of the 7th International Conference on Language and Automata Theory and Applications*, LATA'13, pages 347–358. Springer, 2013.

[HLL14] M. Hutagalung, M. Lange, and E. Lozes. Buffered simulation games for Büchi automata. In *Proceedings 14th International Conference on Automata and Formal Languages*, AFL'14, pages 286–300, 2014.

[HMU06] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[Hol04] G. J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004.

[HU67] J.E. Hopcroft and J.D. Ullman. Nonerasing stack automata. *Journal of Computer and System Sciences*, 1(2):166 – 186, 1967.

[Hut17] M. Hutagalung. Topological characterisation of multi-buffer simulation. In *Proceedings of the 11th International Workshop Reachability Problems*, RP'17, pages 101–117. Springer, 2017.

[Jap94] G. Japaridze. The logic of arithmetical hierarchy. *Annals of Pure and Applied Logic*, 66:89–112, 1994.

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proceeding of the 17th Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[Kec95] A.S. Kechris. *Classical Descriptive Set Theory*. Graduate Texts in Mathematics. Springer-Verlag, 1995.

[KK94] R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[KM02] A. Kucera and R. Mayr. Simulation preorder over simple process algebras. *Information and Computation*, 173(2):184–198, 2002.

[Koc09] C. Koch. Applications of automata in XML processing. In S. Maneth, editor, *Implementation and Application of Automata*, pages 2–2. Springer, 2009.

[Koz97] D. C. Kozen. *Automata and Computability*. Springer-Verlag New York, Inc., 1st edition, 1997.

[LJBA01] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proceeding of the 28th ACM Symposium on Principles of Programming Languages*, pages 81–92. ACM, 2001.

[LK93] C. L. Lucchesi and T. Kowaltowski. Applications of finite automata representing large vocabularies. *Software: Practice and Experience*, 23(1):15–30, 1993.

[Löd14] C. Löding. Decision problems for deterministic pushdown automata on infinite words. In *Proceeding of the 14th International Conference on Automata and Formal Languages*, AFL'14, pages 55–73, 2014.

[Maz77] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. *DAIMI Report Series*, 6(78), 1977.

[Maz89] A. W. Mazurkiewicz. Basic notions of trace theory. In *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354, pages 285–363. Springer, 1989.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proceeding of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489. British Computer Society, 1971.

[Mos80] Y. N. Moschovakis. *Descriptive Set Theory*. Elsevier Science Limited, 1980.

[MP99] A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In M. Kutyłowski, L. Pacholski, and T. Wierzbicki, editors, *Proceeding of the 24th International Symposium on Mathematical Foundations of Computer Science*, MFCS'99, pages 81–91. Springer, 1999.

[MS73] A. R. Meyer and L. J. Stockmeyer. Word problems requiring exponential time. In *Proceeding of the 5th Symposium on Theory of Computing*, STOC'73, pages 1–9. ACM, 1973.

[MSB$^+$16] P. Metzler, H. Saissi, P. Bokor, R. Hesse, and N. Suri. Efficient verification of program fragments: Eager POR. In *Proceeding of the 14th International Symposium Automated Technology for Verification and Analysis*, ATVA'16, pages 375–391, 2016.

[NNH99] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., 1999.

[Pap94] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pos46] E. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 53:264–268, 1946.

[PP04] J.-E. Pin and D. Perrin. *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier, 2004.

[Ram30] F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930.

[RJ87] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.

[Ros81] J. G. Rosenstein. *Linear Orderings*. Pure and applied mathematics. Elsevier, 1981.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[Saf88] S. Safra. On the complexity of omega-automata. In *Proceeding of the 29th Symposium on Foundations of Computer Science*, pages 319–327. IEEE Computer Society, 1988.

[Sak92] J. Sakarovitch. The ”last” decision problem for rational trace languages. In *Proceeding of the 1st Latin American Symposium on Theoretical Informatics*, LATIN'92, pages 460–473. Springer, 1992.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sch07] T. Schwentick. Automata for XML-a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[Sch09] S. Schewe. Büchi complementation made tight. In S. Albers and J.-Y. Marion, editors, *Proceeding of the 26th International Symposium on Theoretical Aspects of Computer Science*, STACS'09, pages 661–672. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2009.

[Sip96] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[Srb06] J. Srba. Visibly pushdown automata: From language equivalence to simulation and bisimulation. In Z. Ésik, editor, *Proceeding of the 20th Internation Workshop on Computer Science Logic*, CSL'06, pages 89–103. Springer, 2006.

[SS78] W. J. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In *Proceeding of the 10th ACM Symposium on Theory of Computing*, pages 275–286. ACM, 1978.

[SVW87] A. P. Sistla, M. Y. Vardi, and F. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2):217 – 237, 1987.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. 1990.

[TPKC07] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *Proceeding of the 14th ACM Conference on Computer and Communications Security*, pages 519–528. ACM, 2007.

[Var96] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, pages 238–266. Springer, 1996.

[vD18] Tom van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, pages 291–308, 2018.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceeding of the 1st Symposium on Logic in Computer Science, LICS'86*, pages 332–344. IEEE, 1986.

[VW94] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[Wan60] H. Wang. Proving theorems by pattern recognition I. *Communications of the ACM*, 3(4):220–234, 1960.

[Yan08] Q. Yan. Lower bounds for complementation of omega-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1), 2008.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 21(2):99–135, 1987.