BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

TWO-WAY FINITE AUTOMATA: AN ALTERNATIVE CONCEPT

ALTERNATIVNÍ KONCEPT DVOUSMĚRNÝCH KONEČNÝCH AUTOMATŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DOMINIK NEJEDLÝ**
AUTOR PRÁCE

SUPERVISOR **prof. RNDr. ALEXANDER MEDUNA, CSc.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

161805

Institut:	Department of Information Systems (DIFS)
Student:	**Nejedlý Dominik, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Mathematical Methods
Title:	**Two-Way Finite Automata: an Alternative Concept**
Category:	Theoretical Computer Science
Academic year:	2024/25

Assignment:

1. Based upon the supervisor's instructions, study selected topics concerning two-way finite automata.
2. Based upon the supervisor's suggestions, introduce new versions of two-way finite automata.
3. Study properties of the automata introduced in 2.
4. Design and implement a software tool that simulates the automata introduced in 2.
5. Test the tool from 4. Evaluate its advantages and disadvantages. Suggest improvements or alternative ways of implementation.
6. Summarize the results. Discuss the future investigation concerning this project.

Literature:

- Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Fazekas, S. Z., Hoshi, K., Yamamura, A.: Two-way deterministic automata with jumping mode, Theoretical Computer Science, Volume 864, 2021. https://doi.org/10.1016/j.tcs.2021.02.030
- Kapoutsis, C., Zakzok, M.: Alternation in two-way finite automata, Theoretical Computer Science, Volume 870, 2021. https://doi.org/10.1016/j.tcs.2020.12.011

Requirements for the semestral defence:
Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Meduna Alexandr, prof. RNDr., CSc.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	21.5.2025
Approval date:	22.10.2024

Abstract

This thesis introduces and studies an alternative concept of two-way finite automata referred to as input-erasing two-way finite automata. Like the original model, these new automata can also move their read heads freely left or right on their input tapes. However, each time they read a symbol, they also erase it from their tapes. The thesis demonstrates that these automata define precisely the family of linear languages and are thus strictly stronger than their original versions. Furthermore, it introduces a variety of restrictions placed upon these automata and the way they work and investigates the effect of these restrictions on their accepting power. In particular, it explores mutual relations between the language families resulting from these restrictions and shows that some of them reduce the power of these automata to that of even linear grammars or even ordinary finite automata.

Abstrakt

Tato práce zavádí a studuje alternativní koncept dvousměrných konečných automatů, který je označován jako vstup vymazávající dvousměrné konečné automaty. Podobně jako původní model mohou i tyto automaty posunovat čtecí hlavu po vstupní pásce libovolně doleva či doprava, avšak každý přečtený symbol ze vstupní pásky vymazávají. Práce demonstruje, že tyto automaty definují přesně třídu lineárních jazyků a jsou tedy silnější než jejich původní verze. Dále zavádí různá omezení kladená na tyto automaty a způsob, kterým pracují, a zkoumá vliv těchto omezení na jejich přijímací sílu. Zabývá se především vzájemnými vztahy mezi jazykovými rodinami, které z těchto omezení vyplývají, a ukazuje, že některá z nich snižují sílu těchto automatů na úroveň vyrovnaných lineárních gramatik nebo dokonce běžných konečných automatů.

Keywords

input-erasing two-way finite automata, linear languages, even linear languages, alternating computation, even computation, left and right moves

Klíčová slova

vstup vymazávající dvousměrné konečné automaty, lineární jazyky, vyrovnané lineární jazyky, střídavý výpočet, vyrovnaný výpočet, levé a pravé přechody

Reference

NEJEDLÝ, Dominik. *Two-Way Finite Automata: an Alternative Concept.* Brno, 2025. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. RNDr. Alexander Meduna, CSc.

Rozšířený abstrakt

Konečné automaty, představené již před více než osmdesáti lety v [28], vždy plnily a plní v informatice mimořádně důležitou roli, a to jak v teorii, tak v praxi. Z toho důvodu bylo definováno mnoho jejich různých variant s cílem poskytnout každé oblasti informatiky tu, která jejím potřebám vyhovuje co možná nejvíce. Jednou z těchto variant jsou dvousměrné konečné automaty, jež se vyznačují tím, že umožňují pohyb čtecí hlavy po vstupní pásce oběma směry, nikoli pouze postupně zleva doprava, jako je tomu u původních (jednosměrných) konečných automatů. Tyto automaty byly nezávisle na sobě představeny v [39] a [43] a od té doby jsou neustále intenzivně zkoumány z různých úhlů pohledu (viz například [2, 3, 4, 13, 21, 22, 47]).

Tato diplomová práce pokračuje v tomto dlouhodobě aktivním výzkumu dvousměrných konečných automatů tím, že zavádí jejich alternativní koncept označovaný jako vstup vymazávající dvousměrné konečné automaty, který disponuje vyšší přijímací sílou než jejich koncept původní. Vzhledem k tomuto navýšení síly pak může být zajímavé, že myšlenka, na níž je tohle jejich nové pojetí založeno, v zásadě vychází z původní koncepce jednosměrných konečných automatů, jež mohou číst každý vstupní symbol pouze jednou. Vstup vymazávající dvousměrné konečné automaty totiž fungují podobně jako klasické dvousměrné konečné automaty, avšak (1) odstraňují již přečtené vstupní symboly, čímž zamezují jejich opětovnému zpracování, a navíc (2) mohou zahájit výpočet z kterékoli pozice na vstupní pásce, nikoli pouze z jejího levého okraje.

Práce nejprve opakuje veškerou terminologii nezbytnou k jejímu ucelenému pochopení. Zahrnuje jak základní pojmy, jako jsou řetězce a jazyky, tak i některé typy formálních gramatik a jednosměrné konečné automaty. Následně popisuje klasický koncept dvousměrných konečných automatů. Uvádí, jakým způsobem fungují, a demonstruje jejich vlastnosti.

Klíčová část této práce pak zavádí a studuje vstup vymazávající dvousměrné konečné automaty. Formálně je definuje, včetně jejich dílčích verzí, poukazuje na jejich stěžejní rozdíly oproti klasickým dvousměrným konečným automatům a ilustruje jejich schopnosti. Hlavním výsledkem je, že tyto nové automaty charakterizují přesně třídu lineárních jazyků, jež plně zahrnuje třídu jazyků regulárních definovanou klasickými jednosměrnými a dvousměrnými konečnými automaty. Toto zjištění je založeno na tom, že libovolný vstup vymazávající dvousměrný konečný automat lze převést na ekvivalentní lineární gramatiku (generující totožný jazyk) a naopak. Navíc je ukázáno, že jednotlivé definované verze těchto automatů jsou stejně silné.

Dále se práce zabývá několika omezeními kladenými na způsob výpočtu vstup vymazávajících dvousměrných konečných automatů, jež jsou založena na střídavém provádění levých a pravých přechodů. Je studován jejich vliv na výpočetní sílu těchto automatů, přičemž jsou stanoveny vzájemné vztahy jazykových rodin z těchto omezení plynoucích.

Nakonec práce zkoumá různá omezení vstupu nově zavedených automatů. Studuje tedy jejich přijímací sílu za předpokladu, že jejich vstupní řetězce či jejich části náleží do jazyků z nějakých předem stanovených jazykových rodin. Je ukázáno, že omezení vstupu založená na regulárních jazycích nevedou k žádnému zvýšení síly těchto automatů. Některá z nich ji dokonce přímo snižují do třídy regulárních jazyků. Oproti tomu však omezení vstupu založená na lineárních jazycích mohou rozšířit jejich přijímací schopnosti i na některé jazyky, které nejsou bezkontextově volné.

Součástí této práce je rovněž implementace programu, který vstup vymazávající dvousměrné konečné automaty simuluje. Jedná se o konzolovou aplikaci implementovanou v jazyce Python, která umožňuje simulovat jejich výpočetně omezené i neomezené běhy a jejímž

hlavním účelem je demonstrovat, jak tyto nové automaty fungují a jak se mohou chovat v praxi.

Two-Way Finite Automata: an Alternative Concept

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. RNDr. Alexander Meduna, CSc. The supplementary information regarding the correctness of the presented formal proofs was provided by Ing. Zbyněk Křivka, Ph.D. I have listed all the literary sources, publications, and other sources that were used during the preparation of this thesis.

.....
Dominik Nejedlý
May 8, 2025

Acknowledgements

I would like to thank my supervisor, prof. RNDr. Alexander Meduna, CSc., for his support and expert guidance during the work on this thesis. I would also like to express my gratitude to Ing. Zbyněk Křivka, Ph.D., for his valuable comments and advice on the formal proofs presented in this thesis. Finally, my heartfelt thanks go to all my loved ones who have always supported me throughout my studies and beyond.

I acknowledge the use of AI tools, specifically ChatGPT and Grammarly, for the sole purpose of grammar and style checking.

Contents

Chapter 1

Introduction

Finite automata, introduced more than eight decades ago in [28], have always fulfilled a crucially important role in computer science both in theory and in practice. Thus, it comes as no surprise that the theory of computation has defined a great variety of these automata in order to provide every computer science area with the version that fits its needs as optimally as possible. Two-way finite automata, independently introduced in [39] and [43], represent significant versions of this kind, which have been constantly and intensively investigated since their introduction from various angles. First of all, in terms of these automata, the theory of computation has studied most of its classical topics, such as nondeterminism, time and space complexity, or purely mathematical properties (see [3, 4, 5, 6, 10, 18, 20, 22, 33, 35, 36, 41]). Furthermore, this theory has introduced various formal models closely related to two-way finite automata in many respects, such as their power or the way they work (see [19, 27]). The theory of computation has also defined and studied several new versions of these automata based upon concepts used in its latest investigation trends, such as the formalization of quantum or jumping computation (see [2, 12, 38, 45, 46, 47]). In addition, apart from the models mentioned above, many other versions of two-way finite automata have been introduced to formalize various features of computation in such terms as probability, alternation, and others (see [13, 14, 21, 25, 40]).

The present thesis continues with this long-time vivid investigation trend by introducing other versions of two-way finite automata, which are, however, stronger than their originals. Indeed, these newly introduced versions characterize the linear language family, which properly contains the regular language family defined by two-way finite automata. Considering this increase in power, it is surprising that the fundamental idea underlying these new versions actually comes from the very original concept of one-way finite automata, which can read every input symbol only once. That is to say, once an input symbol is read, it is also erased, so it cannot be re-read again. To give an insight into these new versions as well as the way they work, we first briefly informally recall the basic notion of a (one-way) finite automaton as well as that of its two-way variant while pointing out the features that have inspired the introduction of the new versions.

The notion of a (one-way) finite automaton describes a machine that, directed by its finite state control, reads an input string on its input tape symbol by symbol in a left-to-right way. Since the entire string is processed in a single pass, reading an occurrence of a symbol can also be considered as its erasure. Indeed, once the occurrence of the symbol is read, it is, in effect, gone as well because the automaton can never re-read this particular occurrence of the symbol during the rest of its computation.

The notion of a two-way finite automaton closely resembles its one-way counterpart. However, as opposed to its strictly left-to-right behavior, the two-way finite automaton can freely move its read head either left or right on its input tape. Consequently, the same occurrence of a symbol can be re-read over and over again, so the automaton never erases it from the tape.

Based upon a combination of the two previous models, we now sketch the new notion of a two-way finite automaton referred to as an *input-erasing two-way finite automaton*. This automaton functions similarly to the classical two-way finite automaton; however, it erases the input symbols just as the one-way finite automaton does and, in addition, can start its computation at any position on the input tape. As a result, despite its ability to move the read head on the tape in both directions, this automaton can never read the same occurrence of a symbol more than once during its computation.

As its fundamental result, this thesis demonstrates that input-erasing two-way finite automata are stronger than one-way or two-way finite automata, which both characterize the regular language family. Indeed, it shows that the input-erasing two-way versions define the linear language family, which properly contains the regular language family. In addition, this thesis discusses two kinds of restrictions placed upon input-erasing two-way finite automata and the way they work. The first kind concerns their computation. More precisely, it restricts the performance of left and right moves in a variety of evenly alternating ways and investigates how these restrictions affect their computational power. The other kind explores input-related restrictions. That is, it studies the power of these automata working under the assumption that their input strings or their parts belong to languages from some prescribed language families, such as the regular and linear language families.

The present thesis is organized as follows. Chapter 2 recalls all the terminology needed in this thesis, covering formal languages and their families, various types of formal grammars, and finite automata. Chapter 3 introduces two-way finite automata, including their accepting capabilities. Chapter 4 describes and formally defines input-erasing two-way finite automata—the new type of two-way finite automata. Chapter 5 presents the fundamental results achieved in this thesis. Namely, it demonstrates that input-erasing two-way finite automata and linear grammars have the same expressive power and that different versions of these automata have the same accepting capabilities. Chapter 6 investigates a variety of evenly alternating restrictions placed upon the way these automata work. Chapter 7 explores the various input-related restrictions of these automata. Chapter 8 closes the present study by summarizing its results and pointing out important open problem areas.

Chapter 2

Preliminaries

This chapter covers the concepts and terminology that are necessary for a comprehensive understanding of this thesis. In particular, it introduces terms such as alphabets, strings, and languages, along with two fundamental models—formal grammars and finite automata—including some of their versions. The reader is assumed to be familiar with the basics of set theory, elementary logic, and the theory of automata and formal languages (see, for instance, [17, 29, 44]). The definitions, conventions, and theorems presented in this chapter are inspired by [15, 23, 29, 30, 32, 42].

Convention 2.1. For any finite set of nonnegative integers X, $\max(X)$ denotes its *maximum*. For any integer n, $\text{abs}(n)$ denotes its *absolute value*. For a set X, $\text{card}(X)$ denotes the *cardinality* of X—the number of members of X.

Definition 2.2. Let X and Y be sets; we say that X and Y are *incomparable* if $X \not\subseteq Y$, $Y \not\subseteq X$, and $X \cap Y \neq \varnothing$.

Convention 2.3. Throughout this thesis, $\mathbb{N}$ denotes the set of *natural numbers*—that is, $\mathbb{N} = \{1, 2, \ldots\}$, and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$.

Convention 2.4. In what follows, we sometimes abbreviate s_1 *if and only if* s_2, where s_1 and s_2 are two *statements*, to s_1 *iff* s_2.

2.1 Strings and Languages

This section introduces strings and languages in terms of formal language theory, including some operations over them.

Definition 2.5. An *alphabet* is a finite, nonempty set of elements, which are called *symbols*.

Definition 2.6. Let Σ be an alphabet. Any finite sequence of symbols from Σ is a *string* over Σ. The string that contains no symbols is called the *empty string* and is denoted by ε. The set of all strings over Σ (including ε) is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Convention 2.7. In what follows, for a string $(a_1, a_2, \ldots, a_n)$, for some $n \in \mathbb{N}$, we write $a_1 a_2 \ldots a_n$ instead of the sequential notation.

Definition 2.8. Let Σ be an alphabet, and let $x = a_1 a_2 \ldots a_n$ be a string over Σ, where $a_i \in \Sigma$, for all $i = 1, 2, \ldots, n$, for some $n \in \mathbb{N}_0$ (the case when $n = 0$ means that $x = \varepsilon$); that is, $x \in \Sigma^*$. The *length* of x, denoted by $|x|$, is defined as $|x| = n$. The *reversal* of

x, denoted by reversal(x), is defined as reversal(x) $= a_n a_{n-1} \dots a_1$. For $a \in \Sigma$, occur(x, a) denotes the number of occurrences of a in x.

Notice that $|\varepsilon| = 0$ and reversal$(\varepsilon) = \varepsilon$.

Definition 2.9. Let Σ be an alphabet, and let $x, y \in \Sigma^*$ be two strings over Σ. The *concatenation* of x and y, denoted by xy, is the string obtained by appending y to x.

Note that for every string x, $\varepsilon x = x\varepsilon = x$.

Definition 2.10. Let Σ be an alphabet, and let $x \in \Sigma^*$ be a string over Σ. For all $n \in \mathbb{N}_0$, the nth *power* of x, denoted by x^n, is recursively defined as

(1) $x^0 = \varepsilon$, and

(2) $x^n = xx^{n-1}$, for $n \geq 1$.

Definition 2.11. Let Σ be an alphabet. Any subset $L \subseteq \Sigma^*$ is a (*formal*) *language* over Σ. L is *finite* if card$(L) = n$, for some $n \in \mathbb{N}$; otherwise, L is *infinite*. L is *singular* if card$(L) = 1$. L is *regular* if it is expressible using a regular expression (see, for instance, Definition 3.23 in [30]).

Definition 2.12. Let Σ be an alphabet, and let $L_1, L_2 \in \Sigma^*$ be two languages over Σ. The *concatenation* of L_1 and L_2, denoted by L_1L_2, is defined as $L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$.

Definition 2.13. Let Σ be an alphabet, and let $L \in \Sigma^*$ be a language over Σ. For all $n \in \mathbb{N}_0$, the nth *power* of L, denoted by L^n, is recursively defined as

(1) $L^0 = \{\varepsilon\}$, and

(2) $L^n = LL^{n-1}$, for $n \geq 1$.

Definition 2.14. Any set whose members are languages is called a *family of languages* (or a *language family*).

Convention 2.15. Let ${}_{sing}\Phi$, ${}_{fin}\Phi$, and ${}_{reg}\Phi$ denote the families of singular, finite, and regular languages, respectively. Furthermore, let ${}_{even}\Phi$ and ${}_{odd}\Phi$ denote the families of languages consisting of even-length and odd-length strings, respectively.

2.2 Grammars

Formal grammars constitute one of the fundamental concepts in formal language theory. Informally, they represent language-generating devices that produce strings by repeatedly rewriting symbols according to some rewriting rules until no symbol can be rewritten. In this section, we introduce several different types of grammars and define the families of languages they generate.

Definition 2.16. A *phrase-structure grammar* (*PSG* for short) is a quadruple

$$G = (N, T, P, S),$$

where

- N is an alphabet of *nonterminals*;

- T is an alphabet of *terminals* such that $N \cap T = \varnothing$;
- $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ is a finite relation called the *set of* (*rewriting*) *rules*;
- $S \in N$ is the *start nonterminal*.

Convention 2.17. In what follows, instead of $(x, y) \in P$, we write $x \to y \in P$.

Definition 2.18. Let $G = (N, T, P, S)$ be a PSG. Over $(N \cup T)^*$, we define the binary *direct derivation relation*, symbolically denoted by $\Rightarrow$, as follows: for all $x \to y \in P$ and $u, v \in (N \cup T)^*$, $uxv \Rightarrow uyv$ in G. In other words, G makes a *derivation step* from uxv to uyv according to a rule of the form $x \to y$. Let $\Rightarrow^n$, for some $n \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the nth power of $\Rightarrow$, the transitive closure of $\Rightarrow$, and the reflexive-transitive closure of $\Rightarrow$, respectively. If $\alpha \Rightarrow^* \beta$ in G, where $\alpha \in (N \cup T)^* N (N \cup T)^*$ and $\beta \in (N \cup T)^*$, we say that G makes a *derivation* from α to β. The *language generated by* G, denoted by $L(G)$, is the set of strings defined as $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. Let $w \in T^*$. We say that G *generates* w if and only if $w \in L(G)$.

Definition 2.19. Let $G = (N, T, P, S)$ be a PSG. G is a *context-sensitive grammar* (*CSG* for short) if each rule in P is of the form $xAy \to xzy$, where $x, y \in (N \cup T)^*$, $A \in N$, and $z \in (N \cup T)^+$. One possible exception is allowed; that is, a rule of the form $S \to \varepsilon$, whose occurrence in P implies that S does not occur on the right-hand side of any rule in P. G is a *context-free grammar* (*CFG* for short) if each rule in P is of the form $A \to x$, where $A \in N$ and $x \in (N \cup T)^*$. G is a *linear grammar* (*LG* for short) if each rule in P is of the form $A \to xBy$ or $A \to x$, where $A, B \in N$ and $x, y \in T^*$. G is a *regular grammar* (*RG* for short) if each rule in P is of the form $A \to x$, where $A \in N$ and $x \in TN \cup T \cup \{\varepsilon\}$.

Definition 2.20. Let $G = (N, T, P, S)$ be an LG. G is an *even linear grammar* (*ELG for short*) if $A \to xBy \in P$, where $A, B \in N$ and $x, y \in T^*$, implies that $|x| = |y|$.

Figure 2.1 visually illustrates a sequence of derivation steps in an LG.

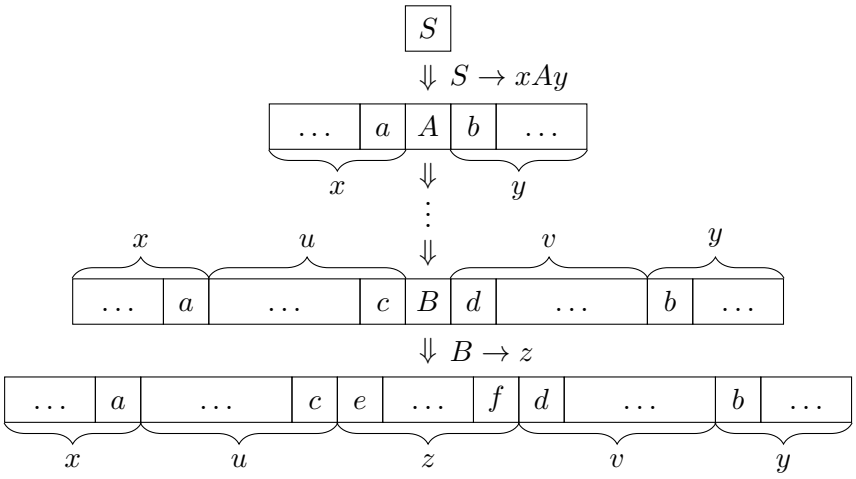

Figure 2.1: A sequence of derivation steps in an LG, where S is the start nonterminal, A and B are nonterminals, x, y, u, v, and z are strings of terminals, $S \to xAy$ and $B \to z$ are rewriting rules, and a, b, c, d, e, and f are terminals.

Next, we give two examples demonstrating how the previously defined grammars work.

Example 2.21. Consider the CSG

$$G = (\{S, A, B\}, \{a, b, c\}, P, S)$$

with the following six rules in P:

$$\begin{array}{lll} S \to \varepsilon, & A \to aABc, & cB \to Bc, \\ S \to A, & A \to abc, & bB \to bb. \end{array}$$

Observe that G generates the language $L(G) = \{a^n b^n c^n \mid n \geq 0\}$. Indeed, first, G rewrites S to A. Then, it generates the same number of as, Bs, and cs by repeatedly replacing A with $aABc$. Thus, all occurrences of a precede all occurrences of B and c. Furthermore, G also replaces each occurrence of cB with Bc to ensure that all occurrences of B precede all occurrences of c. Once G rewrites A to abc, it gradually removes all Bs by successively replacing occurrences of bB with bb; thus, G eliminates all remaining nonterminals and generates $a^i b^i c^i$, for some $i \geq 1$. Note that G can also generate the empty string by simply replacing S with ε at the beginning of a derivation process.

For example, the string $aaabbbccc$ can be generated by G in the following way:

$$\begin{aligned} S \Rightarrow A \Rightarrow aABc \Rightarrow aaABcBc \Rightarrow aaabcBcBc \Rightarrow aaabBccBc \Rightarrow aaabbccBc \\ \Rightarrow aaabbcBcc \Rightarrow aaabbBccc \Rightarrow aaabbbccc. \end{aligned}$$

Recall that $L(G)$ in Example 2.21 is a well-known non-context-free, context-sensitive language (see Example 8.1 in [30]).

Example 2.22. Consider the LG

$$G = (\{S, A\}, \{a, b, c\}, P, S),$$

with P containing the following rules:

$$\begin{array}{ll} S \to Sc, & A \to aAbb, \\ S \to \varepsilon, & A \to \varepsilon, \\ S \to aAbb. & \end{array}$$

G starts every derivation by generating an arbitrary number of cs to the right of S. After this initial phase, G either rewrites S to ε or replaces S with $aAbb$ and continues the derivation by repeatedly generating bb to the right and a to the left of A until A is rewritten to ε. As a result, the languages generated by G is $L(G) = \{a^n b^{2n} c^m \mid m, n \geq 0\}$.

For instance, the string $aabbbbccc$ is generated by G as follows:

$$S \Rightarrow Sc \Rightarrow Scc \Rightarrow Sccc \Rightarrow aAbbccc \Rightarrow aaAbbbbccc \Rightarrow aabbbbccc.$$

Convention 2.23. Let ${}_{CSG}\Phi$, ${}_{CFG}\Phi$, ${}_{LG}\Phi$, ${}_{ELG}\Phi$, and ${}_{RG}\Phi$ denote the families of languages generated by CSGs, CFGs, LGs, ELGs, and RGs, respectively. That is, for all $X \in \{CSG, CFG, LG, ELG, RG\}$, set ${}_{X}\Phi = \{L(G) \mid G \text{ is an X}\}$.

Theorem 2.24. ${}_{sing}\Phi \subset {}_{fin}\Phi \subset {}_{reg}\Phi = {}_{RG}\Phi \subset {}_{ELG}\Phi \subset {}_{LG}\Phi \subset {}_{CFG}\Phi \subset {}_{CSG}\Phi$.

Proof. ${}_{sing}\Phi \subset {}_{fin}\Phi$ is clear from the definitions of the corresponding languages. ${}_{fin}\Phi \subset {}_{reg}\Phi \subset {}_{ELG}\Phi \subset {}_{LG}\Phi \subset {}_{CFG}\Phi \subset {}_{CSG}\Phi$ follows from the Chomsky Hierarchy (see [8, 9]). ${}_{reg}\Phi = {}_{RG}\Phi$ is obvious (see, for instance, Section 7.2 in [29]). ${}_{reg}\Phi \subset {}_{ELG}\Phi$ is established in [1], and ${}_{ELG}\Phi \subset {}_{LG}\Phi$ can be proven using the pumping lemma for even linear languages (see [26]). In fact, $L(G)$ from Example 2.22 is a linear language that cannot be generated by any ELG; that is, $L(G) \in {}_{LG}\Phi \setminus {}_{ELG}\Phi$. □

2.3 Finite Automata

Finite automata are language-recognizing devices that represent one of the fundamental models for regular languages. In this section, we formally introduce (one-way) finite automata, including some of their variants (namely, general, simple, and ε-free versions), and give some of their basic properties.

Conceptually, a (one-way) finite automaton consists of a finite set of states, an input tape, a read head, and a finite state control. The input tape is divided into squares, each of which contains one symbol. On the tape, in a left-to-right manner, the automaton operates by performing a sequence of moves directed by its finite state control. During each of these moves, the automaton changes its current state, reads the current input symbol under its read head, and shifts the read head precisely one square to the right on the input tape. Note that there are also types of finite automata that, during their moves, do not have to read any symbols or can read multiple consecutive symbols at once and shift their read heads accordingly. Notice that since the automaton always shifts its read head to the right, no occurrence of any symbol can be re-read during the rest of the move sequence. The automaton has one state defined as the start state and some states designated as final states. With an input string on the input tape, the automaton starts each computation from the start state with the leftmost symbol of the string under the read head. If it can read the entire input string by a sequence of moves sketched above and, in addition, enter a final state, then the input string is accepted.

A general schema of a finite automaton is shown in Figure 2.2.

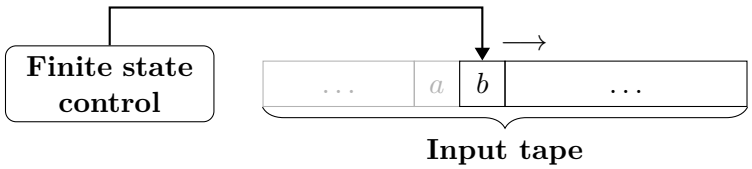

Figure 2.2: General schema of a (one-way) finite automaton.

Definition 2.25. A *general finite automaton* (*GFA* for short) is a quintuple

$$M = (Q, \Sigma, R, s, F),$$

where

- Q is a finite, nonempty set of *states*;
- Σ is an *input alphabet* such that $Q \cap \Sigma = \varnothing$;
- $R \in Q\Sigma^* \times Q$ is a finite relation called the *set of rules*;
- $s \in Q$ is the *start state*;
- $F \subseteq Q$ is the set of *final states*.

Convention 2.26. In what follows, instead of $(\alpha, q) \in R$, we write $\alpha \to q \in R$.

Definition 2.27. Let $M = (Q, \Sigma, R, s, F)$ be a GFA. Over $Q\Sigma^*$, we define the binary *move relation*, symbolically denoted by $\Rightarrow$, as follows: for all $\alpha \to q \in R$ and $u \in \Sigma^*$, $\alpha u \Rightarrow qu$ in M. That is, M makes a *move* (or a *computational step*) from αu to qu

according to a rule of the form $\alpha \to q$. As usual, $\Rightarrow^n$, for some $n \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the nth power of $\Rightarrow$, the transitive closure of $\Rightarrow$, and the reflexive-transitive closure of $\Rightarrow$, respectively. If $\beta \Rightarrow^* \gamma$ in M, where $\beta, \gamma \in Q\Sigma^*$, we say that M makes a *computation* from β to γ. The *language accepted by* M, denoted by $L(M)$, is the set of strings defined as $L(M) = \{w \in \Sigma^* \mid sw \Rightarrow^* f, f \in F\}$. Let $w \in \Sigma^*$. We say that M *accepts* w if and only if $w \in L(M)$; otherwise, M *rejects* w.

Less formally, by applying a rule of the form $px \to q$, where $p, q \in Q$ and $x \in \Sigma^*$, M reads x from its input tape and changes its current state from p to q. In this fashion, then, M reads each input string $w \in \Sigma^*$ sequentially from left to right and accepts it if, starting from s with the leftmost symbol of w being the current input symbol, it ends up in a final state after reading all the symbols of w.

Definition 2.28. Let $M = (Q, \Sigma, R, s, F)$ be a GFA. M is *ε-free* if and only if for all $p, q \in Q$ and $x \in \Sigma^*$, $px \to q \in R$ implies that $|x| \geq 1$. M is said to be a *simple finite automaton* (*SFA* for short) if and only if for all $p, q \in Q$ and $x \in \Sigma^*$, $px \to q \in R$ implies that $|x| \leq 1$.

Figure 2.3 illustrates a GFA move.

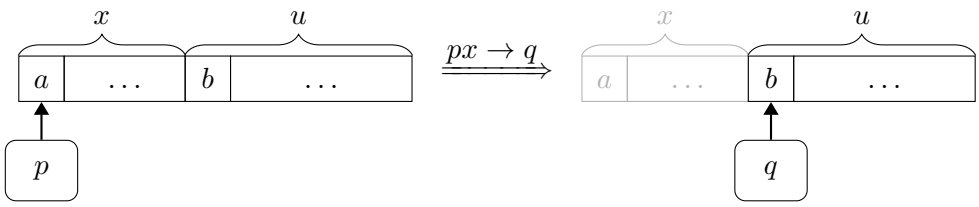

Figure 2.3: A GFA move, where p and q are states, x and u are strings, $px \to q$ is a rule, and a and b are symbols.

Next, we demonstrate the previously defined automata by two examples.

Example 2.29. Consider the ε-free SFA

$$M = (\{s, q, f\}, \{a, b\}, R, s, \{f\})$$

with the following five rules in R (see Figure 2.4):

$$sa \to s, \qquad qa \to q, \qquad fa \to f$$
$$sb \to q, \qquad qb \to f.$$

Starting from s, M reads an arbitrary number of as and then moves from s to q, reading b. In q, like in s, M reads any number of as and then moves from q to f, reading another b. Finally, in f, M reads an arbitrary number of as once again. As a result, the language accepted by M is $L(M) = \{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$.

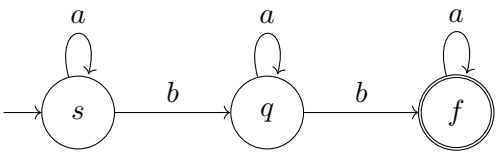

Figure 2.4: State diagram of the ε-free SFA M from Example 2.29.

For example, the string *abaabaaa* is accepted by M as follows:

$$sabaabaaa \Rightarrow sbaabaaa \Rightarrow qaabaaa \Rightarrow qabaaa \Rightarrow qbaaa \Rightarrow faaa \Rightarrow faa \Rightarrow fa \Rightarrow f.$$

Example 2.30. Consider the ε-free SFA

$$M = (\{s, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b\}, R, s, \{s\})$$

with R containing the following rules (see Figure 2.5):

$$\begin{array}{lllllll} sa \to q_1, & q_1a \to q_2, & q_2a \to q_3, & q_3a \to s, & q_4a \to q_5, & q_5a \to q_6, & q_6a \to q_7, \quad q_7a \to q_4 \\ sb \to q_4, & q_1b \to q_5, & q_2b \to q_6, & q_3b \to q_7, & q_4b \to s, & q_5b \to q_1, & q_6b \to q_2, \quad q_7b \to q_3. \end{array}$$

Observe that M accepts the language $L(M) = \{w \in \Sigma^* \mid \text{occur}(w, a) = 4m, \text{occur}(w, b) = 2n, m, n \in \mathbb{N}_0\}$. That is, M accepts only strings over $\{a, b\}$ in which the number of as is a multiple of four and, simultaneously, the number of bs is even. During a computation, each state of M corresponds to a unique combination of remainders when the number of as read so far is divided by four and the number of bs read so far is divided by two. Specifically, the states s, q_1, q_2, and q_3 correspond to the remainders 0, 1, 2, and 3 of the number of as, respectively, when the number of bs is even, and similarly, the states q_4, q_5, q_6, and q_7 correspond to the same remainders of the number of as when the number of bs is odd.

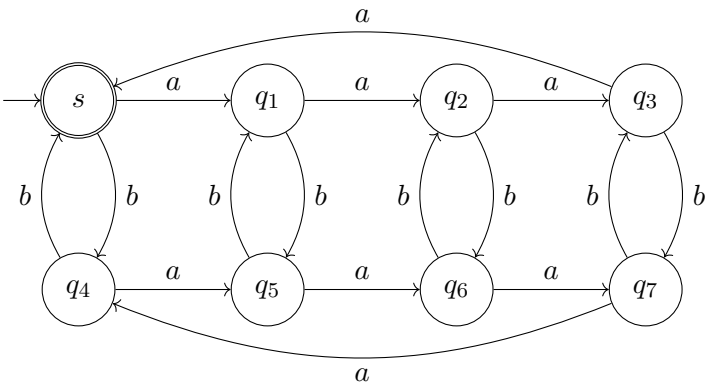

Figure 2.5: State diagram of the ε-free SFA M from Example 2.30.

For instance, M accepts the string *bbabaaba* in the following way:

$$sbbabaaba \Rightarrow q_4babaaba \Rightarrow sabaaba \Rightarrow q_1baaba \Rightarrow q_5aaba \Rightarrow q_6aba \Rightarrow q_7ba \Rightarrow q_3a \Rightarrow s.$$

Convention 2.31. Let ${}_{GFA}^{\varepsilon}\Phi$, ${}_{GFA}\Phi$, ${}_{SFA}^{\varepsilon}\Phi$, and ${}_{SFA}\Phi$ denote the families of languages accepted by GFAs, ε-free GFAs, SFAs, and ε-free SFAs, respectively.

Theorem 2.32 (see [44]). For every GFA M, there is an ε-free SFA M' such that $L(M') = L(M)$. That is, ${}_{GFA}^{\varepsilon}\Phi = {}_{GFA}\Phi = {}_{SFA}^{\varepsilon}\Phi = {}_{SFA}\Phi$.

Theorem 2.33 (see [44]). A language K is regular if and only if there is an ε-free SFA M such that $L(M) = K$. Therefore, ${}_{reg}\Phi = {}_{SFA}\Phi$.

Theorems 2.32 and 2.33 clearly show that finite automata characterize precisely the family of regular languages.

Corollary 2.34. ${}_{reg}\Phi = {}_{GFA}^{\varepsilon}\Phi = {}_{GFA}\Phi = {}_{SFA}^{\varepsilon}\Phi = {}_{SFA}\Phi$.

Chapter 3

Two-Way Finite Automata

The present chapter introduces two-way finite automata, a model based on classical finite automata (see Section 2.3) that extends their behavior by allowing the read head to move both to the left and right on the input tape during the computational process. It formally defines these automata, provides illustrative examples, and presents their accepting power. The information in this chapter is based on [39, 43], with examples inspired by [24, 36].

As mentioned above, a two-way finite automaton works just like a standard (one-way) finite automaton, except that it does not have to operate on its input tape strictly in a left-to-right manner; instead, during each computational step, it can move its read head either to the left or right. This ability to move the read head in both directions allows it to re-read any occurrence of any symbol on the tape arbitrarily many times. Consequently, the automaton can traverse input strings (or their parts) repeatedly, checking their various properties separately. Like its one-way counterpart, the automaton starts each computation from its start state with its read head placed over the leftmost symbol of an input string on its input tape. If it can make a sequence of moves such that it shifts the head off the right end of the tape and, in addition, enters a final state, the input string is accepted.

Figure 3.1 shows a general schema of a two-way finite automaton.

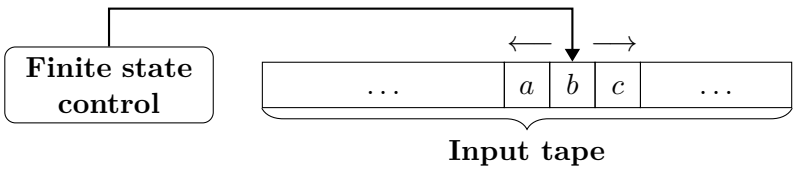

Figure 3.1: General schema of a two-way finite automaton.

Definition 3.1. A *two-way finite automaton* (*2FA* for short) is a quintuple

$$M = (Q, \Sigma, R, s, F),$$

where

- Q is a finite, nonempty set of *states*;
- Σ is an *input alphabet* such that $Q \cap \Sigma = \varnothing$;
- $R \subseteq Q\Sigma \times Q\{\lnot, \rnot\}$ is a finite relation called the *set of rules*, where $\lnot$ and $\rnot$ are two special symbols such that $\{\lnot, \rnot\} \cap (Q \cup \Sigma) = \varnothing$;

- $s \in Q$ is the *start state*;
- $F \subseteq Q$ is the set of *final states*.

Convention 3.2. In what follows, instead of $(\alpha, \beta) \in R$, we write $\alpha \rightarrow \beta \in R$.

Definition 3.3. Let $M = (Q, \Sigma, R, s, F)$ be a 2FA. Over $\Sigma^*Q\Sigma^*$, we define the binary *move relation*, symbolically denoted by $\Rightarrow$, as follows: (i) for all $pa \rightarrow q\Rsh \in R$, where $p, q \in Q$ and $a \in \Sigma$, and $u, v \in \Sigma^*$, $upav \Rightarrow uaqv$ in M, and (ii) for all $pa \rightarrow q\Lsh \in R$, where $p, q \in Q$ and $a \in \Sigma$, $u, v \in \Sigma^*$, and $b \in \Sigma$, $ubpav \Rightarrow uqbav$ in M. That is, (i) M makes a *right move* from $upav$ to $uaqv$ according to a rule of the form $pa \rightarrow q\Rsh$, and similarly, (ii) M makes a *left move* from $ubpav$ to $uqbav$ according to a rule of the form $pa \rightarrow q\Lsh$. As usual, $\Rightarrow^n$, for some $n \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the nth power of $\Rightarrow$, the transitive closure of $\Rightarrow$, and the reflexive-transitive closure of $\Rightarrow$, respectively. If $\alpha \Rightarrow^* \beta$ in M, where $\alpha, \beta \in \Sigma^*Q\Sigma^*$, we say that M makes a *computation* from α to β. The *language accepted by* M, denoted by $L(M)$, is the set of strings defined as $L(M) = \{w \in \Sigma^* \mid sw \Rightarrow^* wf, f \in F\}$. Let $w \in \Sigma^*$. We say that M *accepts* w if and only if $w \in L(M)$; otherwise, M *rejects* w.

Figure 3.2 schematize 2FA moves.

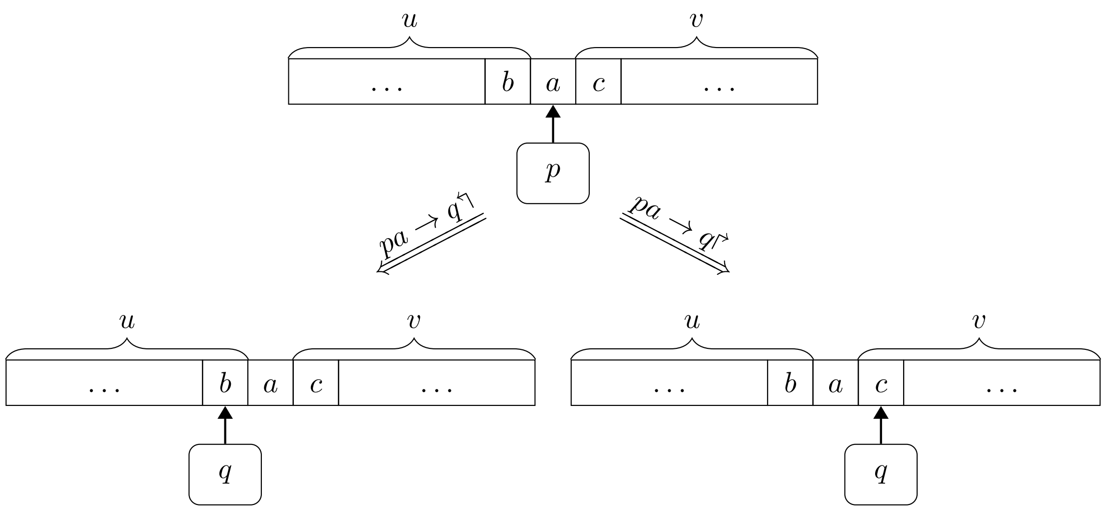

Figure 3.2: A left 2FA move and a right 2FA move, where p and q are states, u and v are strings, $pa \Rightarrow q\Lsh$ and $pa \Rightarrow q\Rsh$ are rules, and a, b, and c are symbols.

Next, we present two examples to illustrate the behavior of these automata.

Example 3.4. Consider the 2FA

$$M = (\{s, q_1, q_2, q_3, q_4, q_5, q_6, f\}, \{a, b, \vdash, \dashv\}, R, s, \{f\})$$

with the following rules in R (see Figure 3.3):

$$\begin{array}{lllll}
s\vdash \rightarrow q_1\Rsh, & q_1\dashv \rightarrow q_2\Lsh, & q_3a \rightarrow q_4\Lsh, & q_5a \rightarrow q_6\Rsh, & q_6b \rightarrow q_6\Rsh \\
q_1a \rightarrow q_1\Rsh, & q_2a \rightarrow q_3\Lsh, & q_3b \rightarrow q_4\Lsh, & q_6a \rightarrow q_6\Rsh, & q_6\dashv \rightarrow f\Rsh, \\
q_1b \rightarrow q_1\Rsh, & q_2b \rightarrow q_3\Lsh, & q_4b \rightarrow q_5\Lsh. & &
\end{array}$$

With a string on its input tape, M first scans the string from left to right and checks that it starts with $\vdash$ (the left end marker of the tape), ends with $\dashv$ (the right end marker

of the tape), and contains only as and bs in between these two symbols. Next, starting with $\dashv$ under its read head, M performs a sequence of left moves to shift the read head three positions back on the tape and then checks that the string contains the sequence ab precisely two symbols before $\dashv$. After this, M finally shifts its read head off $\dashv$, entering the final state f. Clearly, the language accepted by M is $L(M) = \{\vdash\}\{a,b\}^*\{ab\}\{a,b\}^2\{\dashv\}$.

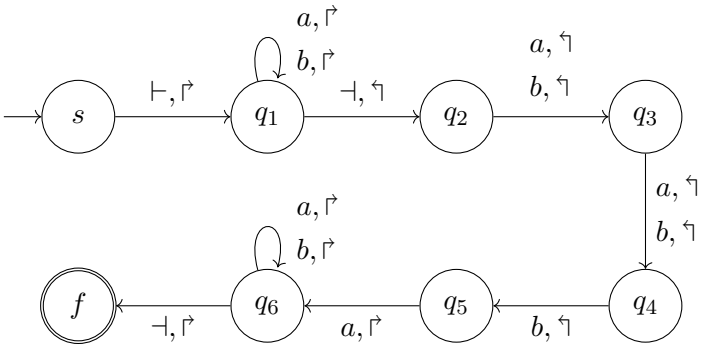

Figure 3.3: State diagram of the 2FA M from example 3.4.

M accepts, for instance, the string $\vdash aabab\dashv$ as follows:

$$\begin{aligned} s\vdash aabab\dashv &\Rightarrow \vdash q_1aabab\dashv \Rightarrow \vdash aq_1abab\dashv \Rightarrow \vdash aaq_1bab\dashv \Rightarrow \vdash aabq_1ab\dashv \Rightarrow \vdash aabaq_1b\dashv \\ &\Rightarrow \vdash aababq_1\dashv \Rightarrow \vdash aabaq_2b\dashv \Rightarrow \vdash aabq_3ab\dashv \Rightarrow \vdash aaq_4bab\dashv \Rightarrow \vdash aq_5abab\dashv \\ &\Rightarrow \vdash aaq_6bab\dashv \Rightarrow \vdash aabq_6ab\dashv \Rightarrow \vdash aabaq_6b\dashv \Rightarrow \vdash aababq_6\dashv \Rightarrow \vdash aabab\dashv f. \end{aligned}$$

Example 3.5. Consider the 2FA

$$M = (\{s, q_1, q_2, q_3, q_4, q_5, q_6, q_7, f\}, \{a, b, \vdash, \dashv\}, R, s, \{f\})$$

with R containing the following rules (see Figure 3.4):

$$\begin{array}{llllll} s\vdash \to q_1{\uparrow}, & q_2b \to q_2{\uparrow}, & q_3a \to q_4{\uparrow}, & q_1\dashv \to q_5{\downarrow}, & q_6a \to q_6{\downarrow}, & q_7a \to q_7{\uparrow}, \\ q_1b \to q_1{\uparrow}, & q_2a \to q_3{\uparrow}, & q_4b \to q_4{\uparrow}, & q_5a \to q_5{\downarrow}, & q_6b \to q_5{\downarrow}, & q_7b \to q_7{\uparrow}, \\ q_1a \to q_2{\uparrow}, & q_3b \to q_3{\uparrow}, & q_4a \to q_1{\uparrow}, & q_5b \to q_6{\downarrow}, & q_5\vdash \to q_7{\uparrow}, & q_7\dashv \to f{\uparrow}. \end{array}$$

M starts every computation by traversing its input tape from left to right, checking that it contains a string consisting only of as and bs and delimited by $\vdash$ on the left and $\dashv$ on the right, and that the number of as in the string is a multiple of four. After this, M continues by scanning the string from right to left to ensure that it also contains an even number of bs. Finally, by a sequence of right moves, M traverses the tape once again, shifts its read head off $\dashv$, and enters f. Hence, M accepts the language $L(M) = \{\vdash\}\{w \in \Sigma^* \mid \text{occur}(w,a) = 4m, \text{occur}(w,b) = 2n, m, n \in \mathbb{N}_0\}\{\dashv\}$.

Consider the string $\vdash abaaba\dashv$. M accepts this string by the following sequence of moves:

$$\begin{aligned} s\vdash abaaba\dashv &\Rightarrow \vdash q_1abaaba\dashv \Rightarrow \vdash aq_2baaba\dashv \Rightarrow \vdash abq_2aaba\dashv \Rightarrow \vdash abaq_3aba\dashv \Rightarrow \vdash abaaq_4ba\dashv \\ &\Rightarrow \vdash abaabq_4a\dashv \Rightarrow \vdash abaabaq_1\dashv \Rightarrow \vdash abaabq_5a\dashv \Rightarrow \vdash abaaq_5ba\dashv \Rightarrow \vdash abaq_6aba\dashv \\ &\Rightarrow \vdash abq_6aaba\dashv \Rightarrow \vdash aq_6baaba\dashv \Rightarrow \vdash q_5abaaba\dashv \Rightarrow q_5\vdash abaaba\dashv \Rightarrow \vdash q_7abaaba\dashv \\ &\Rightarrow \vdash aq_7baaba\dashv \Rightarrow \vdash abq_7aaba\dashv \Rightarrow \vdash abaq_7aba\dashv \Rightarrow \vdash abaaq_7ba\dashv \Rightarrow \vdash abaabq_7a\dashv \\ &\Rightarrow \vdash abaabaq_7\dashv \Rightarrow \vdash abaaba\dashv f. \end{aligned}$$

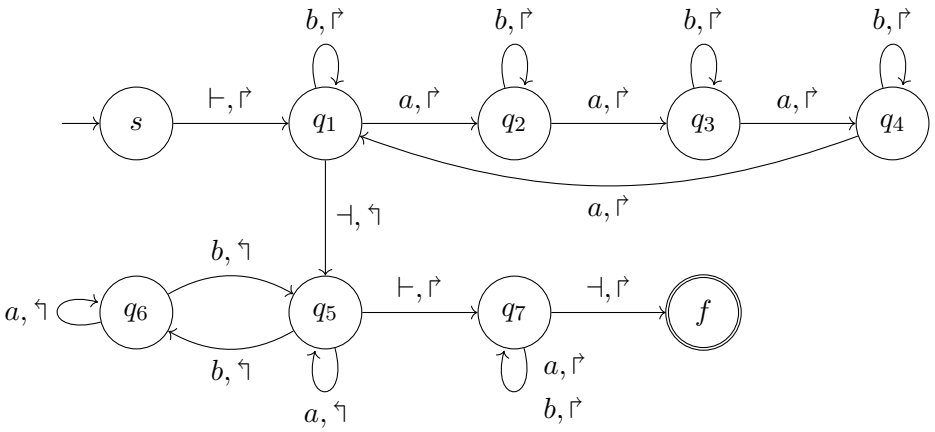

Figure 3.4: State diagram of the 2FA M from example 3.5.

Notice that the language accepted by the 2FA from Example 3.5 is the same, except for the delimitation of its strings by $\vdash$ and $\dashv$, as that accepted by the SFA from Example 2.30. In fact, although 2FAs can be modeled to work in the same way as one-way finite automata, Example 3.5 demonstrates how they can address the same problems differently.

Convention 3.6. Let ${}_{2FA}\Phi$ denote the family of languages accepted by 2FAs. That is, set ${}_{2FA}\Phi = \{L(M) \mid M \text{ is a 2FA}\}$.

As the following theorem states, 2FAs, like classical (one-way) finite automata (see Section 2.3), accept precisely the family of regular languages.

Theorem 3.7 (see Theorem 1 in [43] or Theorem 15 in [39]).

$$ {}_{reg}\Phi = {}_{SFA}\Phi = {}_{2FA}\Phi. $$

Chapter 4

Input-Erasing Two-Way Finite Automata

In this chapter, we introduce input-erasing two-way finite automata, a model based on two-way finite automata (see Chapter 3) that, just like classical (one-way) finite automata (see Section 2.3), does not re-read any symbols on the input tape. Specifically, we formally define the general and simple versions of these automata, including their ε-free alternatives. Just as with classical finite automata, during every move, the former can read a string, which may consist of several symbols, while the latter always reads no more than one input symbol. The ε-free versions behave the same as their originals, except they cannot perform moves without reading any input symbols.

As already mentioned, an input-erasing two-way finite automaton works, in essence, like a two-way finite automaton, except that it erases the input symbols. Indeed, once an occurrence of an input symbol is read on the input tape, it is erased from it (mathematically speaking, this occurrence of the input symbol is changed to the empty string), so the automaton can never re-read it again later during its computation. The automaton starts working on an input string on its input tape from the start state with its read head positioned anywhere within the string. If it can read (and therefore erase) the entire string by a sequence of left, right, or stationary moves and, in addition, enter a final state, it accepts the input string.

A general schema of an input-erasing two-way finite automaton is given in Figure 4.1.

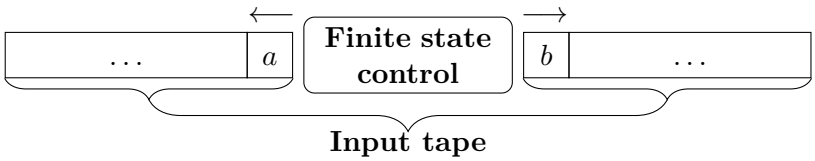

Figure 4.1: General schema of an input-erasing two-way finite automaton.

Definition 4.1. An *input-erasing two-way general finite automaton* (*IE2GFA* for short) is a quintuple

$$M = (Q, \Sigma, R, s, F),$$

where

- Q is a finite, nonempty set of *states*;

- Σ is an *input alphabet* such that $Q \cap \Sigma = \varnothing$;
- $R \subseteq (Q\Sigma^* \cup \Sigma^* Q) \times Q$ is a finite relation called the *set of rules*;
- $s \in Q$ is the *start state*;
- $F \subseteq Q$ is the set of *final states*.

Convention 4.2. In what follows, instead of $(\alpha, q) \in R$, we write $\alpha \to q \in R$.

Definition 4.3. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. Let $\mathcal{C} = \Sigma^* Q \Sigma^*$ be the set of all *configurations* of M. Over $\mathcal{C}$, we define the binary *move relation*, symbolically denoted by $\Rightarrow$, as follows: for all $\alpha \to q \in R$ and $u, v \in \Sigma^*$, $u\alpha v \Rightarrow vqu$ in M. In other words, M makes a *move* (or a *computational step*) from $u\alpha v$ to uqv according to a rule of the form $\alpha \to q$. As usual, $\Rightarrow^n$, for some $n \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$ denote the nth power of $\Rightarrow$, the transitive closure of $\Rightarrow$, and the reflexive-transitive closure of $\Rightarrow$, respectively. If $\beta \Rightarrow^* \gamma$ in M, where $\beta, \gamma \in \mathcal{C}$, we say that M makes a *computation* from β to γ. The *language accepted by* M, denoted by $L(M)$, is the set of strings defined as $L(M) = \{uv \mid u, v \in \Sigma^*, usv \Rightarrow^* f, f \in F\}$. Let $w \in \Sigma^*$. We say that M *accepts* w if and only if $w \in L(M)$; otherwise, M *rejects* w.

Convention 4.4. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA, and let $r \in R$ be a rule of the form $\alpha \to q$. Then, $\mathrm{lhs}(r)$ and $\mathrm{rhs}(r)$ denote α, called the *left-hand side* of r, and q, called the *right-hand side* of r, respectively.

Definition 4.5. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA, and let $r \in R$. If $\mathrm{lhs}(r) = xq$, where $q \in Q$ and $x \in \Sigma^*$, then r is *left*. Analogously, if $\mathrm{lhs}(r) = qx$, where $q \in Q$ and $x \in \Sigma^*$, then r is *right*. A move made by M according to a left rule is a *left move*, and a move made by M according to a right rule is a *right move*. If $|\mathrm{lhs}(r)| \leq 2$, then r is *simple*. If R contains only simple rules, M is said to be an *input-erasing two-way simple finite automaton* (*IE2SFA* for short). If $|\mathrm{lhs}(r)| = 1$, then r is an *ε-rule* (therefore, every ε-rule is simple). If R contains no ε-rules, M is said to be *ε-free*.

Figure 4.2, given below, visually illustrates IE2GFA moves.

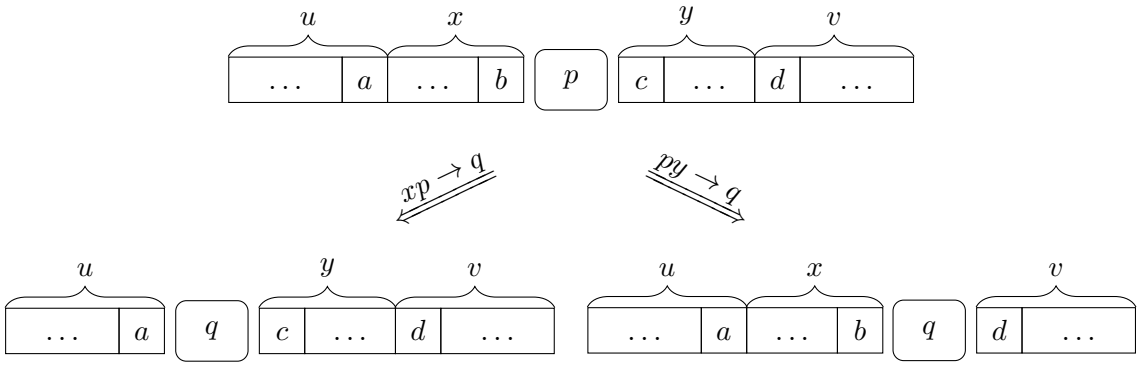

Figure 4.2: A left IE2GFA move and a right IE2GFA move, where p and q are states, u, v, x, and y are strings, $xp \Rightarrow q$ and $py \Rightarrow q$ are rules, and a, b, c, and d are symbols.

Convention 4.6. In what follows, without any loss of generality, we assume that for every IE2GFA $M = (Q, \Sigma, R, s, F)$, $(Q \cup \Sigma) \cap \{\curvearrowleft, \curvearrowright\} = \varnothing$, as we use $\curvearrowleft$ and $\curvearrowright$ to represent move directions (left and right) in state diagrams of IE2GFAs. Note that we do not specify any

direction for any move that M makes according to an ε-rule, as it is a left move and a right move at the same time.

Next, we show two examples demonstrating the behavior of the automata defined here.

Example 4.7. Consider the ε-free IE2SFA

$$M = (\{s, q_1, q_2, q_3, f\}, \{a, b\}, R, s, \{f\}),$$

where R consists of the following rules (see Figure 4.3):

$$\begin{array}{llll} sa \to q_1, & q_2a \to q_3, & q_3a \to f, & af \to f, \\ q_1b \to q_2, & q_2b \to q_3, & q_3b \to f, & bf \to f. \end{array}$$

Starting from s, M first makes two right moves, reading the string ab and entering the state q_2. From q_2, M makes two additional right moves, each of which reads a or b, and together, they bring M to the state f. Finally, M completes its computation with a series of left moves, ensuring that the portion of its input tape to the left of the initial position of its read head contains only as and bs. Therefore, the language accepted by M is $L = \{a, b\}^*\{ab\}\{a, b\}^2$.

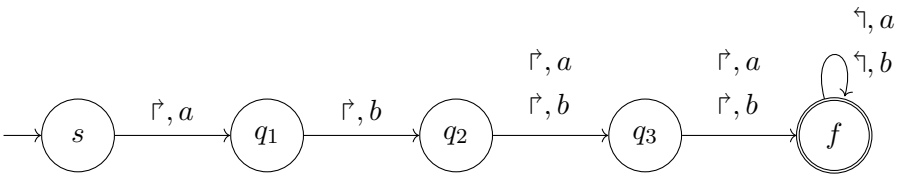

Figure 4.3: State diagram of the ε-free IE2SFA M from Example 4.7.

Consider the string $bbaabaa$. M accepts this string by the following computation:

$$bbasabaa \Rightarrow bbaq_1baa \Rightarrow bbaq_2aa \Rightarrow bbaq_3a \Rightarrow bbaf \Rightarrow bbf \Rightarrow bf \Rightarrow f.$$

Observe that the language accepted by the IE2SFA from Example 4.7 is the same as that accepted by the 2FA from Example 3.4 (except for the delimitation of the strings by $\vdash$ and $\dashv$ in the latter language). In fact, Example 4.7 demonstrates how the ability to erase symbols and start processing the input tape from an arbitrary position could be beneficial for modeling languages with IE2GFAs.

Example 4.8. Consider the IE2SFA

$$M = (\{s, q_1, q_2, f_1, f_2\}, \{a, b, c\}, R, s, \{f_1, f_2\})$$

with R containing the following rules (see Figure 4.4):

$$\begin{array}{llll} sb \to q_1, & s \to f_1, & sc \to q_2, & f_2c \to q_2, \\ aq_1 \to s, & f_1c \to f_1, & aq_2 \to f_2. & \end{array}$$

M starts by repeatedly making two consecutive moves—a right move from s to q_1 that reads b, and a left move from q_1 back to s that reads a. This process allows M to read any string of the form a^ib^i, where $i \geq 0$. After this initial phase, M either changes its current state from s to f_1 without reading any input symbol or makes a right move from s

to q_2, reading c. In f_1, M performs a sequence of right moves, reading an arbitrary number of cs. From q_2, M makes a left move, reading a and entering f_2. Then, M continues by repeatedly making two consecutive moves—a right move from f_2 to q_2 that reads c, and a left move from q_2 back to f_2 that reads a. In this way, M ensures that for every read occurrence of c, an occurrence of a is also read. As a result, M accepts the language $L = \{a^n b^n c^m, a^{m+n} b^n c^m \mid m, n \geq 0\}$.

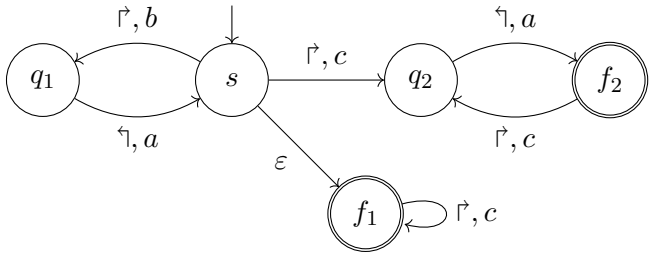

Figure 4.4: State diagram of the IE2SFA M from Example 4.8.

Let *aaaabbcc* be an input string. Using M, it can be accepted as follows:

$$aaaasbbcc \Rightarrow aaaaq_1bcc \Rightarrow aaasbcc \Rightarrow aaaq_1cc \Rightarrow aascc \Rightarrow aaq_2c \Rightarrow af_2c \Rightarrow aq_2 \Rightarrow f_2.$$

Notice that the language $L(M)$ from Example 4.8 is not regular. Specifically, it is a non-regular linear language ($L(M) = {}_{LG}\Phi \setminus {}_{reg}\Phi$). This clearly indicates that IE2GFAs have different accepting power than both GFAs and 2FAs.

Convention 4.9. Let ${}_{IE2GFA}^{\varepsilon}\Phi$, ${}_{IE2GFA}\Phi$, ${}_{IE2SFA}^{\varepsilon}\Phi$, and ${}_{IE2SFA}\Phi$ denote the families of languages accepted by IE2GFAs, ε-free IE2GFAs, IE2SFAs, and ε-free IE2SFAs, respectively.

Chapter 5

Accepting Power

The present chapter demonstrates that IE2GFAs and LGs are equally powerful because they both define the linear language family. Thus, IE2GFAs are stronger than GFAs, which characterize the regular language family, a proper subset of the linear language family. Furthermore, this chapter shows that IE2GFAs and IE2SFAs, along with their ε-free alternatives, possess the same accepting power.

5.1 Equivalence with Linear Grammars

We begin by demonstrating that IE2GFAs and LGs are mutually convertible and thus have the same expressive power.

Lemma 5.1. For every IE2GFA M, there is an LG G such that $L(G) = L(M)$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA M. From M, we next construct an LG $G = (N, T, P, S)$ such that $L(G) = L(M)$. Introduce a new symbol S—the start nonterminal symbol of G. Without any loss of generality, assume that $S \notin Q$. Set $N = Q \cup \{S\}$ and $T = \Sigma$. Initially, set $P = \{s \to \varepsilon\}$. Next, extend P in the following manner:

(1) for each $f \in F$, add $S \to f$ to P;

(2) for each $xq \to p \in R$, where $p, q \in Q$ and $x \in \Sigma^*$, add $p \to xq$ to P;

(3) for each $qx \to p \in R$, where $p, q \in Q$ and $x \in \Sigma^*$, add $p \to qx$ to P;

Basic Idea. G simulates any computation of M in reverse. It starts from the generation of a final state (see step (1)). After this initial derivation step, G simulates every left move made by M according to a rule of the form $xq \to p$, where $p, q \in Q$ and $x \in \Sigma^*$, by using a rule of the form $p \to xq$ (see step (2)). The right moves are simulated analogously (see step (3)). This simulation process is completed by using $s \to \varepsilon$, thus erasing the start state s in order to get a string of terminal symbols in G.

In order to demonstrate $L(G) = L(M)$ rigorously, we first establish the following claim.

Claim 5.1.A. For all $u, v \in \Sigma^*$ and $p, q \in Q$,

$$q \Rightarrow^* upv \text{ in } G \text{ iff } upv \Rightarrow^* q \text{ in } M.$$

Proof of Claim 5.1.A. First, we establish the *only-if* part of this equivalence. That is, by induction on the number of derivation steps $i \geq 0$, we show that $q \Rightarrow^i upv$ in G implies $upv \Rightarrow^* q$ in M.

Basis. Let $i = 0$, so $q \Rightarrow^0 upv$ in G. Then, $q = p$ and $uv = \varepsilon$. Since $q \Rightarrow^0 q$ in M, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all derivations consisting of no more than j steps, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any derivation of the form $q \Rightarrow^{j+1} upv$ in G. Let this derivation start with the application of a rule of the form

$$q \to xo$$

from P, where $o \in Q$ and $x \in \Sigma^*$. Recall that $Q = N \setminus \{S\}$, and observe that S cannot occur on the right-hand side of any rule. Thus, we can express $q \Rightarrow^{j+1} upv$ as

$$q \Rightarrow xo \Rightarrow^j xu'pv$$

in G, where $xu' = u$. Then, by the induction hypothesis, $u'pv \Rightarrow^* o$ in M. As described above, step (2) constructs $q \to xo \in P$ from $xo \to q \in R$, so

$$xu'pv \Rightarrow^* xo \Rightarrow q$$

in M. Because $xu' = u$, $upv \Rightarrow^* q$ in M.

In the case that the derivation $q \Rightarrow^{j+1} upv$ in G starts with the application of a rule of the form $q \to ox$ from P, where $o \in Q$ and $x \in \Sigma^*$, proceed by analogy.

Thus, the induction step is completed.

Next, we establish the *if* part of the equivalence stated in Claim 5.1.A, so we prove that $upv \Rightarrow^i q$ in M implies $q \Rightarrow^* upv$ in G by induction on the number of moves $i \geq 0$.

Basis. For $i = 0$, $upv \Rightarrow^0 q$ occurs in M only for $p = q$ and $uv = \varepsilon$. Clearly, $q \Rightarrow^0 q$ in G. Therefore, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves, for some $j \in \mathbb{N}_0$.

Induction Step. Let $upv \Rightarrow^{j+1} q$ in M, and let this computation end with the application of a rule of the form

$$xo \to q$$

from R, where $o \in Q$ and $x \in \Sigma^*$. Now, we express $upv \Rightarrow^{j+1} q$ as

$$xu'pv \Rightarrow^j xo \Rightarrow q$$

in M, where $xu' = u$. By the induction hypothesis, $o \Rightarrow^* u'pv$ in G. From $xo \to q \in R$, step (2) above constructs $q \to xo \in P$. Thus, G makes

$$q \Rightarrow xo \Rightarrow^* xu'pv$$

with $u = xu'$.

If the computation $upv \Rightarrow^{j+1} q$ in M ends with the application of a rule of the form $ox \to q$ from R, where $o \in Q$ and $x \in \Sigma^*$, proceed analogously.

Thus, the induction step is completed, and Claim 5.1.A holds.

Considering Claim 5.1.A for $p = s$, we see that for all $u, v \in \Sigma^*$ and $q \in Q$, $q \Rightarrow^* usv$ in G iff $usv \Rightarrow^* q$ in M. As follows from the construction technique presented above, G starts every derivation by applying a rule of the form $S \to f$, where $f \in F$, and ends it by applying a rule of the form $s \to \varepsilon$. Consequently, $S \Rightarrow f \Rightarrow^* usv \Rightarrow uv$ in G iff $usv \Rightarrow^* f$ in M, so $L(G) = L(M)$. Thus, Lemma 5.1 holds. □

To illustrate the technique from the proof of Lemma 5.1, we give the following example.

Example 5.2. Consider the IE2SFA M from Example 4.8. Recall that

$$M = (\{s, q_1, q_2, f_1, f_2\}, \{a, b, c\}, R, s, \{f_1, f_2\}),$$

where $R = \{sb \to q_1, aq_1 \to s, s \to f_1, f_1c \to f_1, sc \to q_2, aq_2 \to f_2, f_2c \to q_2\}$, and $L(M) = \{a^n b^n c^m, a^{m+n} b^n c^m \mid m, n \geq 0\}$. From M, the construction technique described in the proof of Lemma 5.1 produces the LG

$$G = (\{S, s, q_1, q_2, f_1, f_2\}, \{a, b, c\}, P, S)$$

with P consisting of the following rules:

$$\begin{array}{lllll} S \to f_1, & f_1 \to s, & f_2 \to aq_2, & q_2 \to sc, & q_1 \to sb, \\ S \to f_2, & f_1 \to f_1c, & q_2 \to f_2c, & s \to aq_1, & s \to \varepsilon. \end{array}$$

G first rewrites S to either f_1 or f_2. From f_1, it generates an arbitrary number of cs to the right by repeatedly rewriting f_1 to f_1c. After that, it replaces f_1 with s, thus generating sc^i, for some $i \geq 0$. From f_2, it generates $a^j sc^j$, for some $j \geq 1$. This is done by repeatedly replacing f_2 with aq_2 and q_2 with f_2c and then rewriting q_2 to sc. After generating either sc^i or $a^j sc^j$, G continues by repeatedly replacing s with aq_1 and q_1 with sb. Thus, from s, it generates $a^k sb^k$, for some $k \geq 0$. Finally, G rewrites s to ε. Clearly, G generates the language $L(G) = \{a^n b^n c^m, a^{m+n} b^n c^m \mid m, n \geq 0\}$. Hence, $L(G) = L(M)$.

Observe that G exhibits behavior that is completely inverse to that of M. For instance, the string $aabbcc$ that M accepts by the computation

$$aasbbcc \Rightarrow aaq_1bcc \Rightarrow asbcc \Rightarrow aq_1cc \Rightarrow scc \Rightarrow f_1cc \Rightarrow f_1c \Rightarrow f_1$$

is derived by G as follows:

$$S \Rightarrow f_1 \Rightarrow f_1c \Rightarrow f_1cc \Rightarrow scc \Rightarrow aq_1cc \Rightarrow asbcc \Rightarrow aaq_1bcc \Rightarrow aasbbcc \Rightarrow aabbcc.$$

Lemma 5.3. For every LG G, there is an IE2GFA M such that $L(M) = L(G)$.

Proof. Let $G = (N, T, P, S)$ be an LG. From G, we next construct an IE2GFA $M = (Q, \Sigma, R, s, F)$ such that $L(M) = L(G)$. Introduce a new symbol s—the start state of M. Set $Q' = \{\langle A \to xBy \rangle \mid A \to xBy \in P, A, B \in N, x, y \in T^*\}$. Without any loss of generality, assume that $Q' \cap N = \varnothing$ and $s \notin Q' \cup N$. Set $Q = Q' \cup N \cup \{s\}$, $\Sigma = T$, and $F = \{S\}$. R is constructed as follows:

(1) for each $A \to x \in P$, where $A \in N$ and $x \in T^*$, add $sx \to A$ to R;

(2) for each $A \to xBy \in P$, where $A, B \in N$ and $x, y \in T^*$, add $xB \to \langle A \to xBy \rangle$ and $\langle A \to xBy \rangle y \to A$ to R.

Basic Idea. M simulates any derivation of G in reverse. It starts by reading a string of terminals generated by G in the last step of a derivation (see step (1)). After this initial computational step, M simulates every derivation step made by G according to a rule of the form $A \to xBy$, where $A, B \in N$ and $x, y \in T^*$, by using two consecutive rules of the forms $xB \to \langle A \to xBy \rangle$ and $\langle A \to xBy \rangle y \to A$, where $\langle A \to xBy \rangle$ is a newly introduced state with the rule record to which it relates (see step (2)). The entire simulation process is completed by reaching the state S, which represents the start nonterminal symbol of G, and emptying the input tape.

To establish $L(M) = L(G)$ formally, we first prove the following claim.

Claim 5.3.A. For all $u, v \in T^*$ and $A, B \in N$,

$$uBv \Rightarrow^* A \text{ in } M \text{ iff } A \Rightarrow^* uBv \text{ in } G.$$

Proof of Claim 5.3.A. First, we establish the *only-if* part of this equivalence. By induction on the number of moves $i \geq 0$, we prove that $uBv \Rightarrow^i A$ in M implies $A \Rightarrow^* uBv$ in G.

Basis. Let $i = 0$, so $uBv \Rightarrow^0 A$ in M. Then, $A = B$ and $uv = \varepsilon$. Clearly, $A \Rightarrow^0 A$ in G. For $i = 1$, $uBv \Rightarrow A$ never occurs in M for any $u, v \in T^*$, since, by the construction technique described above, M does not have any rule of the form $xB \to A$ or $By \to A$ for any $x, y \in T^*$. Recall that $A, B \in (Q \setminus Q') \setminus \{s\} = N$, so no rules added by step (1) can be applied here. Hence, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $uBv \Rightarrow^{j+2} A$ in M. Let this computation start with the application of two consecutive rules of the forms

$$xB \to \langle C \to xBy \rangle \text{ and } \langle C \to xBy \rangle y \to C$$

from R, where $C \in N$, $x, y \in T^*$, and $C \to xBy \in P$. Thus, we can express $uBv \Rightarrow^{j+2} A$ as

$$u'xByv' \Rightarrow u'\langle C \to xBy \rangle yv' \Rightarrow u'Cv' \Rightarrow^j A$$

in M, where $u'x = u$ and $yv' = v$. Clearly, by the induction hypothesis, $A \Rightarrow^* u'Cv'$ in G. Step (2) constructs $xB \to \langle C \to xBy \rangle, \langle C \to xBy \rangle y \to C \in R$ from $C \to xBy \in P$, so

$$A \Rightarrow^* u'Cv' \Rightarrow u'xByv'$$

in G. Because $u'x = u$ and $yv' = v$, $A \Rightarrow^* uBv$ in G, and the induction step is completed.

Next, we establish the *if* part of the equivalence stated in Claim 5.3.A, so we show that $A \Rightarrow^i uBv$ in G implies $uBv \Rightarrow^* A$ in M by induction on the number of derivation steps $i \geq 0$.

Basis. For $i = 0$, $A \Rightarrow^0 uBv$ occurs in G only for $A = B$ and $uv = \varepsilon$. Since $A \Rightarrow^0 A$ in M, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all derivations consisting of no more than j steps, for some $j \in \mathbb{N}_0$.

Induction Step. Let $A \Rightarrow^{j+1} uBv$ in G, and let this derivation end with the application of a rule of the form

$$C \to xBy$$

from P, where $C \in N$ and $x, y \in T^*$. Now, we express $A \Rightarrow^{j+1} uBv$ as

$$A \Rightarrow^j u'Cv' \Rightarrow u'xByv'$$

in G, where $u'x = u$ and $yv' = v$. Hence, by the induction hypothesis, $u'Cv' \Rightarrow^* A$ in M. From $C \to xBy \in P$, step (2) above constructs $xB \to \langle C \to xBy \rangle, \langle C \to xBy \rangle y \to C \in R$. Thus, M makes

$$u'xByv' \Rightarrow u'\langle C \to xBy \rangle yv' \Rightarrow u'Cv' \Rightarrow^* A$$

with $u'x = u$ and $yv' = v$, so $uBv \Rightarrow^* A$ in M. Thus, the induction step is completed. Therefore, Claim 5.3.A holds.

Considering Claim 5.3.A for $A = S$, we have that for all $u, v \in T^*$ and $B \in N$, $uBv \Rightarrow^* S$ in M iff $S \Rightarrow^* uBv$ in G. As follows from the above construction technique, M starts every computation by applying a rule of the form $sz \to C$, where $C \in N$ and $z \in T^*$, constructed from $C \to z \in P$, and S is the only final state of M. Consequently, $uszv \Rightarrow uCv \Rightarrow^* S$ in M iff $S \Rightarrow^* uCv \Rightarrow uzv$ in G, so $L(M) = L(G)$. Hence, Lemma 5.3 holds. □

The following example demonstrates the technique used in the proof of Lemma 5.3.

Example 5.4. Return to the LG G from Example 2.22. Recall that

$$G = (\{S, A\}, \{a, b, c\}, P, S),$$

where $P = \{S \to Sc, S \to \varepsilon, S \to aAbb, A \to aAbb, A \to \varepsilon\}$, and $L(G) = \{a^n b^{2n} c^m \mid m, n \geq 0\}$. By applying the technique from the proof of Lemma 5.3 to G, we construct the IE2GFA

$$M = (\{s, S, A, \langle S \to Sc\rangle, \langle S \to aAbb\rangle, \langle A \to aAbb\rangle\}, \{a, b, c\}, R, s, \{S\})$$

with the following rules in R (see Figure 5.1):

$$\begin{array}{llll} s \to S, & S \to \langle S \to Sc\rangle, & aA \to \langle A \to aAbb\rangle, & aA \to \langle S \to aAbb\rangle, \\ s \to A, & \langle S \to Sc\rangle c \to S, & \langle A \to aAbb\rangle bb \to A, & \langle S \to aAbb\rangle bb \to S. \end{array}$$

M starts each computation by moving from s to either A or S without reading any input symbols. From A, it can arbitrarily many times perform two consecutive moves—a left move from A to $\langle A \to aAbb\rangle$ that reads a, and a right move from $\langle A \to aAbb\rangle$ back to A that reads bb. After that, M makes a left move from A to $\langle S \to aAbb\rangle$, reading a, and then a right move from $\langle S \to aAbb\rangle$ to S, reading bb. Finally, by cycling between S and $\langle S \to Sc\rangle$, M can read an arbitrary number of cs to the right. Hence, the language accepted by M is $L(M) = \{a^n b^{2n} c^m \mid m, n \geq 0\}$, and thus $L(M) = L(G)$.

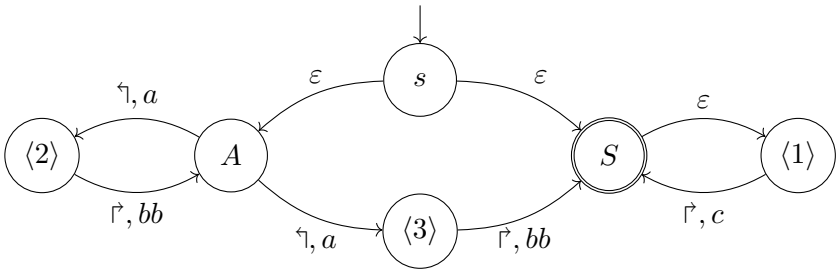

Figure 5.1: State diagram of the IE2GFA M from Example 5.4, where the labels 1, 2, and 3 stand for $S \to Sc$, $A \to aAbb$, and $S \to aAbb$, respectively.

Observe that M works in a completely inverse way to G. For instance, consider the string $aaabbbbbbcc$, which G generates by the derivation

$$S \Rightarrow Sc \Rightarrow Scc \Rightarrow aAbbcc \Rightarrow aaAbbbbcc \Rightarrow aaaAbbbbbbcc \Rightarrow aaabbbbbbcc.$$

M accepts this string by the computation

$$\begin{aligned} aaasbbbbbbcc &\Rightarrow aaaAbbbbbbcc \Rightarrow aa\langle A \to aAbb\rangle bbbbbbcc \Rightarrow aaAbbbbcc \\ &\Rightarrow a\langle A \to aAbb\rangle bbbbcc \Rightarrow aAbbcc \Rightarrow \langle S \to aAbb\rangle bbcc \\ &\Rightarrow Scc \Rightarrow \langle S \to Sc\rangle cc \Rightarrow Sc \Rightarrow \langle S \to Sc\rangle c \Rightarrow S. \end{aligned}$$

Theorem 5.5. ${}_{IE2GFA}^{\varepsilon}\Phi = {}_{LG}^{\varepsilon}\Phi$.

Proof. The inclusion ${}_{IE2GFA}^{\varepsilon}\Phi \subseteq {}_{LG}^{\varepsilon}\Phi$ follows from Lemma 5.1. The opposite inclusion, ${}_{LG}^{\varepsilon}\Phi \subseteq {}_{IE2GFA}^{\varepsilon}\Phi$, follows from Lemma 5.3, so the theorem holds. □

5.2 Equivalence of Variants of Input-Erasing Two-Way Finite Automata

In this section, we show that IE2GFAs, ε-free IE2GFAs, IE2SFAs, and ε-free IE2SFAs have the same accepting power.

Theorem 5.6. ${}_{IE2GFA}^{\varepsilon}\Phi = {}_{IE2SFA}^{\varepsilon}\Phi = {}_{IE2SFA}\Phi$.

Proof. As every IE2SFA is a special case of an IE2GFA, we have ${}_{IE2SFA}^{\varepsilon}\Phi \subseteq {}_{IE2GFA}^{\varepsilon}\Phi$. To prove ${}_{IE2GFA}^{\varepsilon}\Phi \subseteq {}_{IE2SFA}^{\varepsilon}\Phi$, consider any IE2GFA M. From M, we construct an equivalent IE2SFA M' based upon the following idea. Let M read an n-symbol string, $a_1 \dots a_n$, to the right during a single move. M' simulates this move as follows:

(1) M' records $a_1 \dots a_n$ into its current state,

(2) M' makes n subsequent right moves during which it reads $a_1 \dots a_n$ symbol by symbol, proceeding from a_1 towards a_n.

The left moves in M are simulated by M' analogously. The details are left to the reader. Thus, ${}_{IE2GFA}^{\varepsilon}\Phi \subseteq {}_{IE2SFA}^{\varepsilon}\Phi$, and ${}_{IE2GFA}^{\varepsilon}\Phi = {}_{IE2SFA}^{\varepsilon}\Phi$ holds.

As is obvious, ${}_{IE2SFA}\Phi \subseteq {}_{IE2SFA}^{\varepsilon}\Phi$. The opposite inclusion can be established straightforwardly using the standard technique for removing ε-rules (see, for instance, Section 3.2.1 in [30]). Consequently, ${}_{IE2SFA}^{\varepsilon}\Phi = {}_{IE2SFA}\Phi$, and Theorem 5.6 holds. □

As every ε-free IE2SFA is also an ε-free IE2GFA, we clearly obtain the following corollary from the previous theorem.

Corollary 5.7. ${}_{IE2GFA}^{\varepsilon}\Phi = {}_{IE2GFA}\Phi = {}_{IE2SFA}^{\varepsilon}\Phi = {}_{IE2SFA}\Phi$.

Chapter 6

Computational Restrictions

In this chapter, we introduce a variety of restrictions that require the performance of left and right moves in an alternating way, and we investigate how these restrictions affect the computational power of IE2GFAs and IE2SFAs. First, we formally define these computational restrictions. Then, we establish some relations between their corresponding language families, as well as relations between these restricted language families and their original unrestricted variants. Finally, we show that under one of these restrictions, IE2GFAs possess the same expressive power as ELGs.

6.1 Definitions and Examples

In this section, we formally define the computational restrictions of input-erasing two-way finite automata and illustrate them with an example.

Definition 6.1. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA, and let $\mathcal{C} = \Sigma^* Q \Sigma^*$ be the set of all configurations over M. Let $\alpha \Rightarrow^* \beta$ in M, where $\alpha, \beta \in \mathcal{C}$. If, in $\alpha \Rightarrow^* \beta$, every sequence of two consecutive moves satisfies the condition that the first of these two moves reads symbols in one direction while the second move reads symbols in the opposite direction; more precisely, if for every two consecutive moves, i and j, in $\alpha \Rightarrow^* \beta$, i is left if and only if j is right, then $\alpha \Rightarrow^* \beta$ is *alternating*, symbolically written as $\alpha \Rightarrow^*_{alt} \beta$.

Let $\alpha \Rightarrow^*_{alt} \beta$ in M consist of n moves, for some even $n \geq 0$, where $\alpha, \beta \in K$.

(1) If, in $\alpha \Rightarrow^*_{alt} \beta$, for each odd i such that $0 \leq i \leq n$, both the ith and the $(i+1)$th moves read the same number of input symbols, then $\alpha \Rightarrow^*_{alt} \beta$ is an *even computation*, symbolically written as $\alpha \Rightarrow^*_{even} \beta$.

(2) If $\gamma \in \mathcal{C}$, $\gamma \Rightarrow \alpha$ in M, and $\alpha \Rightarrow^*_{even} \beta$ in M, then $\gamma \Rightarrow \alpha \Rightarrow^*_{even} \beta$ is an *initialized even computation*, symbolically written as $\gamma \Rightarrow^*_{init\text{-}even} \beta$.

The *languages accepted by M using alternating computation*, *even computation*, and *initialized even computation* are defined as follows:

$$L(M)_{alt} = \{uv \mid u, v \in \Sigma^*, usv \Rightarrow^*_{alt} f, f \in F\},$$
$$L(M)_{even} = \{uv \mid u, v \in \Sigma^*, usv \Rightarrow^*_{even} f, f \in F\},$$
$$L(M)_{init\text{-}even} = \{uv \mid u, v \in \Sigma^*, usv \Rightarrow^*_{init\text{-}even} f, f \in F\}.$$

To illustrate the previous definition, we give the following example.

Example 6.2. Consider the IE2GFA

$$M = (\{s, q_1, q_2, q_3, f\}, \{a, b, c, d, e\}, R, s, \{s, f\}),$$

where R contains the following rules (see Figure 6.1):

$sa \to q_1$,	$sb \to q_2$,	$sc \to q_3$,	$s \to f$,
$q_1a \to q_1$,	$q_2b \to q_2$,	$q_3c \to q_3$,	$df \to f$,
$aq_1 \to s$,	$bq_2 \to s$,	$cq_3 \to s$,	$fee \to f$.

Without any computational restrictions placed upon it, M first reads, for each occurrence of a, b, or c that it reads to the left, a nonempty sequence of consecutive occurrences of that exact same symbol (a, b, or c) to the right. After that, M continues by reading an arbitrary number of ds to the left and any even number of es to the right. Hence, the language accepted by M is $L(M) = \{d\}^*\{x_1 \dots x_m x_m^{n_m} \dots x_1^{n_1} \mid x_1, \dots, x_m \in \{a, b, c\}, m \geq 0, n_1, \dots, n_m \geq 1\}\{ee\}^*$.

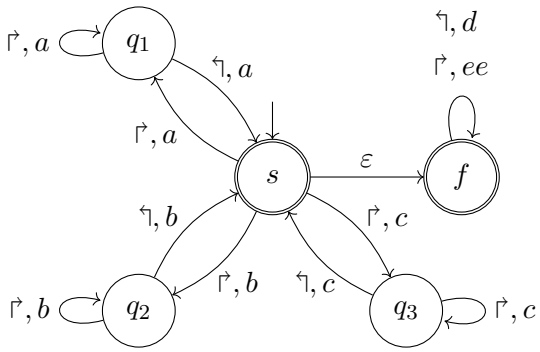

Figure 6.1: State diagram of the IE2GFA M from Example 6.2.

For instance, consider the string $w = dcbaaabcccee$. M can accept w by the following sequence of moves:

$$dcbasaabcccee \Rightarrow dcbaq_1abcccee \Rightarrow dcbaq_1bcccee \Rightarrow dcbsbcccee \Rightarrow dcbq_2cccee \Rightarrow dcscccee$$
$$\Rightarrow dcq_3ccee \Rightarrow dcq_3cee \Rightarrow dcq_3ee \Rightarrow dsee \Rightarrow dfee \Rightarrow df \Rightarrow f.$$

Now, suppose that M uses alternating computation. Under this restriction, M rejects w, as it can never make two consecutive moves in the same direction. In fact, the language accepted by M in this way is $L(M)_{alt} = \{d^m w\, \text{reversal}(w) e^{2n}, d^n e^{2m} \mid w \in \{a, b, c\}^*, 0 \leq n \leq m, m - n \leq 1\}$. Observe that for any string of the form $d^m w\, \text{reversal}(w) e^{2n}$, where $0 \leq n \leq m$, $m - n \leq 1$, and $w \in \{a, b, c\}^+$, the move according to $s \to f \in R$ always acts as a right move. For any string of the form $d^{n-1}e^{2n}$, where $n \geq 1$, that exact same move always acts as a left move, and for any other string in $L(M)_{alt}$, it can act either as a left move or a right move depending on the performed accepting computation on the string.

Next, assume that M works under even computation. Compared to alternating computation, this further restricts the behavior of M so that it can never accept any input in f, regardless of its form. Consequently, this restriction reduces the language accepted by M to $L(M)_{even} = \{w\, \text{reversal}(w) \mid w \in \{a, b, c\}^*\}$.

Finally, consider M operating under initialized even computation. In this case, starting from s, M moves to f without reading any symbols (thus, ε can be accepted) or to q_1, q_2,

or q_3, reading a, b, or c to the right, respectively. In q_1, q_2, and q_3, M must continue by reading the corresponding symbol to the right again without changing its state; otherwise, it cannot reach a final state at the end of any computation of this type. After that, M moves back to s, reading the corresponding symbol to the left, and then continues to work the same way as under even computation. This behavior of M results in the language $L(M)_{\mathit{init\text{-}even}} = \{wx \,\mathrm{reversal}(w) \mid x \in \{aaa, bbb, ccc\}, w \in \{a, b, c\}^*\} \cup \{\varepsilon\}$.

Convention 6.3. Let ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{alt}}$, ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{even}}$, ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{init\text{-}even}}$, ${}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$, ${}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{even}}$, and ${}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{init\text{-}even}}$ denote the families of languages accepted by IE2GFAs and IE2SFAs using alternating computation, even computation, and initialized even computation, respectively. Analogously, let ${}_{\mathit{IE2GFA}}\Phi_{\mathit{alt}}$, ${}_{\mathit{IE2GFA}}\Phi_{\mathit{even}}$, ${}_{\mathit{IE2GFA}}\Phi_{\mathit{init\text{-}even}}$, ${}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}}$, ${}_{\mathit{IE2SFA}}\Phi_{\mathit{even}}$, and ${}_{\mathit{IE2SFA}}\Phi_{\mathit{init\text{-}even}}$ denote the corresponding families of languages in terms of ε-free versions of these automata.

6.2 Effect on Accepting Power

In this section, we investigate the effect of the previously defined restrictions on the accepting power of IE2GFAs and their variants.

Lemma 6.4. For every IE2GFA M, there is an IE2GFA M' such that $L(M')_{\mathit{alt}} = L(M') = L(M)$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. From M, we construct the IE2GFA

$$M' = (Q, \Sigma, R \cup \{q \to q \mid q \in Q\}, s, F).$$

Clearly, $L(M') = L(M)$. Observe, however, that between every two consecutive moves that M can make, M' can always perform an additional move that does not read any symbols (and thus acts as both a left move and a right move at the same time). This allows M' to simulate any computation of M as an alternating one. Hence, $L(M')_{\mathit{alt}} = L(M)$. □

Theorem 6.5. ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi = {}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{alt}} = {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$.

Proof. The inclusions ${}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}} \subseteq {}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{alt}} \subseteq {}_{\mathit{IE2GFA}}^{\varepsilon}\Phi$ follow directly from the definition of an IE2SFA and the definition of alternating computation (see Definitions 4.5 and 6.1). The opposite inclusions, ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi \subseteq {}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{alt}} \subseteq {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$, follow from Theorem 5.6 and Lemma 6.4. □

Theorem 6.6. ${}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}} \subset {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$.

Proof (Basic Idea). Clearly, ${}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}} \subseteq {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$. To demonstrate that ${}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}} \subset {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$, consider $L = \{a^n b^n c^m \mid n, m \geq 0\}$. Clearly, $L \in {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}}$. Next, we sketch how to prove $L \notin {}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}}$ by contradiction. Assume that there exists an ε-free IE2SFA M such that $L(M)_{\mathit{alt}} = L$. Take any $a^i b^i c^j$, for some $i, j \geq 0$. M has to start its successful computation in between as and bs in order to verify the same number of occurrences of those symbols. After this verification, M has to read the remaining j cs to the right. However, this reading cannot be performed by M working under alternating computation. Thus, $L \in {}_{\mathit{IE2SFA}}^{\varepsilon}\Phi_{\mathit{alt}} \setminus {}_{\mathit{IE2SFA}}\Phi_{\mathit{alt}}$, so Theorem 6.6 holds. □

Theorem 6.7. ${}_{\mathit{IE2GFA}}^{\varepsilon}\Phi_{\mathit{even}} \subset {}_{\mathit{even}}\Phi$.

Proof. As each even computation consists of an even number of moves, each language in ${}_{IE2GFA}^{\varepsilon}\Phi_{even}$ clearly contains only even-length strings. Thus, ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \subset {}_{even}\Phi$. □

Theorem 6.8. ${}_{IE2GFA}^{\varepsilon}\Phi_{even}$ is incomparable with any of these language families—${}_{sing}\Phi$, ${}_{fin}\Phi$, and ${}_{reg}\Phi$.

Proof. Let $L \in {}_{IE2GFA}^{\varepsilon}\Phi_{even}$. By Theorem 6.7, $x \in L$ implies that $|x|$ is even. Thus, any $\{y\} \in {}_{sing}\Phi$ with $|y|$ being odd, such as $\{a\}$, is outside of ${}_{IE2GFA}^{\varepsilon}\Phi_{even}$. Clearly, $\{aa\} \in {}_{IE2GFA}^{\varepsilon}\Phi_{even} \cap {}_{sing}\Phi$. Notice that $\{a^n b^n \mid n \geq 0\} \in {}_{IE2GFA}^{\varepsilon}\Phi_{even} \setminus {}_{sing}\Phi$. The rest of this proof is left to the reader, as it follows the same reasoning. □

Lemma 6.9. For every IE2GFA M, there exists an ε-free IE2SFA M' such that $L(M') = L(M')_{even} = L(M)_{even}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. From M, we next construct an ε-free IE2SFA $M' = (Q', \Sigma, R', s', F')$ such that $L(M') = L(M')_{even} = L(M)_{even}$. Introduce a new symbol s'—the start state of M'. Let $k = \max\{|\text{lhs}(r)| - 1 \mid r \in R\}$. Set $\hat{Q} = \{\langle xqy^{\curvearrowleft}\rangle, \langle yqx^{\curvearrowright}\rangle \mid q \in Q, x, y \in \Sigma^*, |x| + |y| \leq 2k - 1, 0 \leq |y| - |x| \leq 1\}$. Without any loss of generality, assume that $s' \notin \hat{Q}$. Set $Q' = \hat{Q} \cup \{s'\}$. Initially, set $R' = \varnothing$ and $F' = \{\langle f^{\curvearrowleft}\rangle, \langle f^{\curvearrowright}\rangle \mid f \in F\}$. If $s \in F$, add s' to F'. Extend R' by performing steps (1) through (4), given next, until no more rules can be added to R'.

(1) If $a_1 \ldots a_n p \to q, q a_{n+1} \ldots a_{2n} \to o \in R$, where $p, q, o \in Q$ and $a_i \in \Sigma$, $1 \leq i \leq 2n$, for some $n \geq 1$, extend R' by adding

$$\begin{aligned}
a_n\langle p^{\curvearrowleft}\rangle &\to \langle a_1 \ldots a_{n-1} o a_{n+1} \ldots a_{2n}{}^{\curvearrowleft}\rangle,\\
\langle a_1 \ldots a_{n-1} o a_{n+1} \ldots a_{2n}{}^{\curvearrowleft}\rangle a_{n+1} &\to \langle a_1 \ldots a_{n-1} o a_{n+2} \ldots a_{2n}{}^{\curvearrowleft}\rangle,\\
a_{n-1}\langle a_1 \ldots a_{n-1} o a_{n+2} \ldots a_{2n}{}^{\curvearrowleft}\rangle &\to \langle a_1 \ldots a_{n-2} o a_{n+2} \ldots a_{2n}{}^{\curvearrowleft}\rangle,\\
&\vdots\\
\langle o a_{2n}{}^{\curvearrowleft}\rangle a_{2n} &\to \langle o^{\curvearrowleft}\rangle.
\end{aligned}$$

In addition, if $p = s$, also include $a_n s' \to \langle a_1 \ldots a_{n-1} o a_{n+1} \ldots a_{2n}{}^{\curvearrowleft}\rangle$ in R'.

(2) If $p a_n \ldots a_1 \to q, a_{2n} \ldots a_{n+1} q \to o \in R$, where $p, q, o \in Q$ and $a_i \in \Sigma$, $1 \leq i \leq 2n$, for some $n \geq 1$, extend R' by adding

$$\begin{aligned}
\langle p^{\curvearrowright}\rangle a_n &\to \langle a_{2n} \ldots a_{n+1} o a_{n-1} \ldots a_1{}^{\curvearrowright}\rangle,\\
a_{n+1}\langle a_{2n} \ldots a_{n+1} o a_{n-1} \ldots a_1{}^{\curvearrowright}\rangle &\to \langle a_{2n} \ldots a_{n+2} o a_{n-1} \ldots a_1{}^{\curvearrowright}\rangle,\\
\langle a_{2n} \ldots a_{n+2} o a_{n-1} \ldots a_1{}^{\curvearrowright}\rangle a_{n-1} &\to \langle a_{2n} \ldots a_{n+2} o a_{n-2} \ldots a_1{}^{\curvearrowright}\rangle,\\
&\vdots\\
a_{2n}\langle a_{2n} o^{\curvearrowright}\rangle &\to \langle o^{\curvearrowright}\rangle.
\end{aligned}$$

In addition, if $p = s$, add $s' a_n \to \langle a_{2n} \ldots a_{n+1} o a_{n-1} \ldots a_1{}^{\curvearrowright}\rangle$ to R', too.

(3) For each $p \to q, q \to o \in R$ and $r' \in R'$, where $o, p, q \in Q$ and $\text{lhs}(r') = a\langle o^{\curvearrowleft}\rangle$, for some $a \in \Sigma$, add $a\langle p^{\curvearrowleft}\rangle \to \text{rhs}(r')$ to R'; in addition, if $p = s$, add $as' \to \text{rhs}(r')$ to R', too.

(4) For each $p \to q, q \to o \in R$ and $r' \in R'$, where $o, p, q \in Q$ and $\text{lhs}(r') = \langle o^{\curvearrowright}\rangle a$, for some $a \in \Sigma$, add $\langle p^{\curvearrowright}\rangle a \to \text{rhs}(r')$ to R'; moreover, if $p = s$, add $s'a \to \text{rhs}(r')$ to R' as well.

Repeat the following extension of F' until no more states can be included in F'.

(5) For each $p \to q, q \to o \in R$, where $o, p, q \in Q$ and $\langle o^{\curvearrowleft}\rangle, \langle o^{\curvearrowright}\rangle \in F'$, add $\langle p^{\curvearrowleft}\rangle$ and $\langle p^{\curvearrowright}\rangle$ to F'; in addition, if $p = s$, also add s' to F'.

Basic Idea. As is obvious, M' represents an ε-free IE2SFA. M' simulates any even computation in M by making sequences of moves, each of which reads (and thereby erases) a single symbol. To explain step (1), assume that M performs a two-move even computation by rules $a_1 \dots a_n p \to q, q a_{n+1} \dots a_{2n} \to o \in R$, where $o, p, q \in Q$ and $a_i \in \Sigma$, $1 \leq i \leq 2n$, for some $n \geq 1$. Consider the sequence of rules introduced into R' in step (1). Observe that once M' applies its first rule, it has to apply all the remaining rules of this sequence in an uninterrupted one-by-one way, and thereby, it simulates the two-move computation in M. Notice that the first rule, $a_n\langle p^{\curvearrowleft}\rangle \to \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n}{}^{\curvearrowleft}\rangle$, is a left rule. Step (2) is analogous to step (1), except that the first rule of the introduced sequence is a right rule. To explain step (3), assume that (i) M performs an even computation according to two ε-rules $p \to q, q \to o \in R$, where $o, p, q \in Q$, and that (ii) R' contains r' with $\text{lhs}(r') = a\langle o^{\curvearrowleft}\rangle$ (r' is introduced into R' in step (1) or (3)). Then, this step introduces $a\langle p^{\curvearrowleft}\rangle \to \text{rhs}(r')$ into R'. By using this newly introduced rule, $a\langle p^{\curvearrowleft}\rangle \to \text{rhs}(r')$, M' actually skips over the two-move even computation according to $p \to q$ and $q \to o$ in M, after which it enters the state $\text{rhs}(r')$, which occurs as the right-hand side of the first rule of a rule sequence introduced in step (1). Step (4) parallels step (3), except that r' is a right rule in step (4), while it is a left rule in step (3).

Consider F'. Assume that an accepting even computation in M ends with an even sequence of moves according to ε-rules (including the empty sequence). Observe that at this point, by the extension of F' in step (5), M' accepts, too.

To establish $L(M')_{even} = L(M)_{even}$ formally, we first prove the following two claims.

Claim 6.9.A. When M is ε-free, for all $u, v \in \Sigma^*$ and $p, q \in Q$,

$$u\langle p^{\curvearrowleft}\rangle v \Rightarrow^*_{even} \langle q^{\curvearrowleft}\rangle \text{ in } M' \text{ iff } upv \Rightarrow^*_{even} q \text{ in } M,$$

where $upv \Rightarrow^*_{even} q$ starts with a left move (unless it consists of no moves).

Proof of Claim 6.9.A. We begin by proving the *only-if* part of this equivalence. That is, by induction on the number of moves $i \geq 0$, we show that for ε-free M, $u\langle p^{\curvearrowleft}\rangle v \Rightarrow^i_{even} \langle q^{\curvearrowleft}\rangle$ in M' implies that there is $upv \Rightarrow^*_{even} q$ in M that starts with a left move (or consists of no moves at all).

Basis. Let $i = 0$, so $u\langle p^{\curvearrowleft}\rangle v \Rightarrow^0_{even} \langle q^{\curvearrowleft}\rangle$ in M'. Then, $p = q$ and $uv = \varepsilon$. Clearly, $q \Rightarrow^0_{even} q$ in M. For $i = 1$, $u\langle p^{\curvearrowleft}\rangle v \Rightarrow^1_{even} \langle q^{\curvearrowleft}\rangle$ never occurs in M', since, by Definition 6.1, each even computation is supposed to have an even number of moves; however, $u\langle p^{\curvearrowleft}\rangle v \Rightarrow^1_{even} \langle q^{\curvearrowleft}\rangle$ has one move. Thus, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M', for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $u\langle p^{\curvearrowleft}\rangle v \Rightarrow^{j+2n}_{even} \langle q^{\curvearrowleft}\rangle$ in M', for some $n \geq 1$. Let this computation start with the application of $2n$ consecutive rules of the forms

$$a_n\langle p^{\curvearrowleft}\rangle \to \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n}{}^{\curvearrowleft}\rangle,$$

$$\langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} {}^{\curvearrowleft}\rangle a_{n+1} \to \langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} {}^{\curvearrowleft}\rangle,$$
$$a_{n-1}\langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} {}^{\curvearrowleft}\rangle \to \langle a_1 \dots a_{n-2} o a_{n+2} \dots a_{2n} {}^{\curvearrowleft}\rangle,$$
$$\vdots$$
$$\langle o a_{2n} {}^{\curvearrowleft}\rangle a_{2n} \to \langle o {}^{\curvearrowleft}\rangle$$

from R', where $o \in Q$ and $a_k \in \Sigma$ for all $1 \leq k \leq 2n$. Thus, we can express $u\langle p {}^{\curvearrowleft}\rangle v \Rightarrow_{even}^{j+2n} \langle q {}^{\curvearrowleft}\rangle$ as

$$\begin{aligned} u' a_1 \dots a_n \langle p {}^{\curvearrowleft}\rangle a_{n+1} \dots a_{2n} v' &\Rightarrow u' a_1 \dots a_{n-1} \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} {}^{\curvearrowleft}\rangle a_{n+1} \dots a_{2n} v' \\ &\Rightarrow u' a_1 \dots a_{n-1} \langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} {}^{\curvearrowleft}\rangle a_{n+2} \dots a_{2n} v' \\ &\Rightarrow u' a_1 \dots a_{n-2} \langle a_1 \dots a_{n-2} o a_{n+2} \dots a_{2n} {}^{\curvearrowleft}\rangle a_{n+2} \dots a_{2n} v' \\ &\vdots \\ &\Rightarrow u' \langle o a_{2n} {}^{\curvearrowleft}\rangle a_{2n} v' \Rightarrow u' \langle o {}^{\curvearrowleft}\rangle v' \Rightarrow_{even}^{j} \langle q {}^{\curvearrowleft}\rangle \end{aligned}$$

in M', where $u' a_1 \dots a_n = u$ and $a_{n+1} \dots a_{2n} v' = v$. According to the induction hypothesis, $u' o v' \Rightarrow_{even}^{*} q$ in M, and this computation starts with a left move (or consists of no moves). Step (1) above constructs $a_n \langle p {}^{\curvearrowleft}\rangle \to \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} {}^{\curvearrowleft}\rangle, \dots, \langle o a_{2n} {}^{\curvearrowleft}\rangle a_{2n} \to \langle o {}^{\curvearrowleft}\rangle \in R'$ from $a_1 \dots a_n p \to t, t a_{n+1} \dots a_{2n} \to o \in R$, for some $t \in Q$, so M makes

$$u' a_1 \dots a_n p a_{n+1} \dots a_{2n} v' \Rightarrow u' t a_{n+1} \dots a_{2n} v' \Rightarrow u' o v' \Rightarrow_{even}^{*} q.$$

Taking into account the properties of the computation $u' o v' \Rightarrow_{even}^{*} q$, since $u' a_1 \dots a_n = u$ and $a_{n+1} \dots a_{2n} v' = v$, it follows that $upv \Rightarrow_{even}^{*} q$ in M. As we can see, $upv \Rightarrow_{even}^{*} q$ starts with a left move, which completes the induction step.

Next, we prove the *if* part of the equivalence stated in Claim 6.9.A. By induction on the number of moves $i \geq 0$, we show that for ε-free M, if there is $upv \Rightarrow_{even}^{i} q$ in M that starts with a left move (or consists of no moves), then $u\langle p {}^{\curvearrowleft}\rangle v \Rightarrow_{even}^{*} \langle q {}^{\curvearrowleft}\rangle$ in M'.

Basis. Let $i = 0$, so $upv \Rightarrow_{even}^{0} q$ in M. Then, $p = q$ and $uv = \varepsilon$. Clearly, $\langle q {}^{\curvearrowleft}\rangle \Rightarrow_{even}^{0} \langle q {}^{\curvearrowleft}\rangle$ in M'. For $i = 1$, $upv \Rightarrow_{even}^{1} q$ never occurs in M, since, by the definition of even computation (see Definition 6.1), every $upv \Rightarrow_{even}^{*} q$ consists of an even number of moves; however, $upv \Rightarrow_{even}^{1} q$ consists of a single move. Thus, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $upv \Rightarrow_{even}^{j+2} q$ in M. Let this computation start with the application of two consecutive rules of the forms

$$a_1 \dots a_n p \to t \text{ and } t a_{n+1} \dots a_{2n} \to o$$

from R, where $o, t \in Q$ and $a_k \in \Sigma$ for all $1 \leq k \leq 2n$, for some $n \geq 1$. Hence, we can express $upv \Rightarrow_{even}^{j+2} q$ as

$$u' a_1 \dots a_n p a_{n+1} \dots a_{2n} v' \Rightarrow u' t a_{n+1} \dots a_{2n} v' \Rightarrow u' o v' \Rightarrow_{even}^{j} q$$

in M, where $u' a_1 \dots a_n = u$ and $a_{n+1} \dots a_{2n} v' = v$. According to the induction hypothesis, $u' \langle o {}^{\curvearrowleft}\rangle v' \Rightarrow_{even}^{*} \langle q {}^{\curvearrowleft}\rangle$ in M'. From $a_1 \dots a_n p \to t, t a_{n+1} \dots a_{2n} \to o \in R$, step (1) constructs

$$a_n \langle p {}^{\curvearrowleft}\rangle \to \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} {}^{\curvearrowleft}\rangle,$$

$$\langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} \curvearrowleft \rangle a_{n+1} \to \langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} \curvearrowleft \rangle,$$
$$a_{n-1} \langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} \curvearrowleft \rangle \to \langle a_1 \dots a_{n-2} o a_{n+2} \dots a_{2n} \curvearrowleft \rangle,$$
$$\vdots$$
$$\langle o a_{2n} \curvearrowleft \rangle a_{2n} \to \langle o \curvearrowleft \rangle \in R',$$

so M' makes

$$\begin{aligned} u' a_1 \dots a_n \langle p \curvearrowleft \rangle a_{n+1} \dots a_{2n} v' &\Rightarrow u' a_1 \dots a_{n-1} \langle a_1 \dots a_{n-1} o a_{n+1} \dots a_{2n} \curvearrowleft \rangle a_{n+1} \dots a_{2n} v' \\ &\Rightarrow u' a_1 \dots a_{n-1} \langle a_1 \dots a_{n-1} o a_{n+2} \dots a_{2n} \curvearrowleft \rangle a_{n+2} \dots a_{2n} v' \\ &\Rightarrow u' a_1 \dots a_{n-2} \langle a_1 \dots a_{n-2} o a_{n+2} \dots a_{2n} \curvearrowleft \rangle a_{n+2} \dots a_{2n} v' \\ &\vdots \\ &\Rightarrow u' \langle o a_{2n} \curvearrowleft \rangle a_{2n} v' \Rightarrow u' \langle o \curvearrowleft \rangle v' \Rightarrow^*_{even} \langle q \curvearrowleft \rangle. \end{aligned}$$

Notice that by the construction technique of M', $u' \langle o \curvearrowleft \rangle v' \Rightarrow^*_{even} \langle q \curvearrowleft \rangle$ can never start with a right move. This, together with the fact that $u' a_1 \dots a_n = u$ and $a_{n+1} \dots a_{2n} v' = v$, implies that $u \langle p \curvearrowleft \rangle v \Rightarrow^*_{even} \langle q \curvearrowleft \rangle$ in M'. Thus, the induction step is completed, and Claim 6.9.A holds.

Claim 6.9.B. When M is ε-free, for all $u, v \in \Sigma^*$ and $p, q \in Q$,

$$u \langle p \curvearrowright \rangle v \Rightarrow^*_{even} \langle q \curvearrowright \rangle \text{ in } M' \text{ iff } upv \Rightarrow^*_{even} q \text{ in } M,$$

where $upv \Rightarrow^*_{even} q$ starts with a right move (unless it consists of no moves).

Proof of Claim 6.9.B. Prove this claim by analogy with the proof of Claim 6.9.A.

Claims 6.9.A and 6.9.B demonstrate the correctness of steps (1) and (2) from the above construction technique. However, they do not address the elimination of ε-rules of M in steps (3), (4), and (5). For this reason, we next establish Claims 6.9.C, 6.9.D, and 6.9.E.

Claim 6.9.C. For all $x \in \Sigma^*, y \in \Sigma^+, a \in \Sigma$, and $p, t \in Q$ such that $|x| + 1 = |y|$,

$$a \langle p \curvearrowleft \rangle \Rightarrow \langle xty \curvearrowleft \rangle \text{ in } M' \text{ iff there are } o, q \in Q \text{ such that } xapy \Rightarrow^*_{even} xaqy \Rightarrow oy \Rightarrow t \text{ in } M.$$

Proof of Claim 6.9.C. First, we establish the *only-if* part of this equivalence. By induction on the number of iterations of step (3) $i \geq 0$, we show that $a \langle p \curvearrowleft \rangle \Rightarrow \langle xty \curvearrowleft \rangle$ in M' implies that there are $o, q \in Q$ such that $xapy \Rightarrow^*_{even} xaqy \Rightarrow oy \Rightarrow t$ in M.

Basis. For $i = 0$, $a \langle p \curvearrowleft \rangle \Rightarrow \langle xty \curvearrowleft \rangle$ in M' can only be performed using a rule of the form $a \langle p \curvearrowleft \rangle \to \langle xty \curvearrowleft \rangle$ added to R' in step (1). Then, since step (1) constructs $a \langle p \curvearrowleft \rangle \to \langle xty \curvearrowleft \rangle \in R'$ from $xap \to g, gy \to t \in R$, for some $g \in Q$, it follows that $xapy \Rightarrow gy \Rightarrow t$ in M. Thus, the basis holds true.

Induction Hypothesis. Assume that the implication holds for no more than j iterations of step (3), for some $j \in \mathbb{N}_0$.

Induction Step. Consider any $a \langle p \curvearrowleft \rangle \Rightarrow \langle xty \curvearrowleft \rangle$ in M' performed using a rule of the form $a \langle p \curvearrowleft \rangle \to \langle xty \curvearrowleft \rangle$ that belongs to R' from the $(j+1)$th iteration of step (3). From this, it follows that there exist $p \to g, g \to h \in R$, for some $g, h \in Q$, and $a \langle h \curvearrowleft \rangle \to \langle xty \curvearrowleft \rangle \in R'$ that was added to R' during the jth iteration of step (3). Then, by the induction hypothesis, there are $o, q \in Q$ such that $xahy \Rightarrow^*_{even} xaqy \Rightarrow oy \Rightarrow t$ in M, so M can make

$$xapy \Rightarrow xagy \Rightarrow xahy \Rightarrow^*_{even} xaqy \Rightarrow oy \Rightarrow t.$$

By the definition of even computation, $xapy \Rightarrow^*_{even} xaqy \Rightarrow oy \Rightarrow t$ in M. Hence, the induction step is completed.

Next, we establish the *if* part of the equivalence stated in Claim 6.9.C. By induction on the number of moves $i \geq 0$, we show that $xapy \Rightarrow^{i}_{even} xaqy \Rightarrow oy \Rightarrow t$ in M implies $a\langle p\curvearrowleft\rangle \Rightarrow \langle xty\curvearrowleft\rangle$ in M'.

Basis. Let $i = 0$, so $xapy \Rightarrow^{0}_{even} xaqy \Rightarrow oy \Rightarrow t$ in M. Then, $p = q$. Clearly, according to step (1), $a\langle q\curvearrowleft\rangle \to \langle xty\curvearrowleft\rangle \in R'$, so $a\langle q\curvearrowleft\rangle \Rightarrow \langle xty\curvearrowleft\rangle$ in M'. Let $i = 1$, so $xapy \Rightarrow^{1}_{even} xaqy \Rightarrow oy \Rightarrow t$ in M. This can never happen because, by Definition 6.1, every even computation consists of an even number of moves; however, $p \Rightarrow^{1}_{even} q$ consists of one move, which is an odd number of moves. Thus, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations of the form $xapy \Rightarrow^{k}_{even} xaqy \Rightarrow oy \Rightarrow t$ in M with $0 \leq k \leq j$, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $xapy \Rightarrow^{j+2}_{even} xaqy \Rightarrow oy \Rightarrow t$ in M, and let it start with the application of two consecutive rules of the forms

$$p \to g \text{ and } g \to h$$

from R, where $g, h \in Q$. Then, we can express $xapy \Rightarrow^{j+2}_{even} xaqy \Rightarrow oy \Rightarrow t$ as

$$xapy \Rightarrow xagy \Rightarrow xahy \Rightarrow^{j}_{even} xaqy \Rightarrow oy \Rightarrow t$$

in M. Clearly, by the induction hypothesis, $a\langle h\curvearrowleft\rangle \Rightarrow \langle xty\curvearrowleft\rangle$ in M'. Hence, $a\langle h\curvearrowleft\rangle \to \langle xty\curvearrowleft\rangle \in R'$. From $a\langle h\curvearrowleft\rangle \to \langle xty\curvearrowleft\rangle \in R'$ and $p \to g, g \to h \in R$, step (3) constructs $a\langle p\curvearrowleft\rangle \to \langle xty\curvearrowleft\rangle \in R'$, so $a\langle p\curvearrowleft\rangle \Rightarrow \langle xty\curvearrowleft\rangle$ in M'. Thus, the induction step is completed, and Claim 6.9.C holds.

Claim 6.9.D. For all $x \in \Sigma^+, y \in \Sigma^*, a \in \Sigma$, and $p, t \in Q$ such that $|x| = |y| + 1$,

$$\langle p\curvearrowright\rangle a \to \langle xty\curvearrowright\rangle \in R' \text{ iff there are } o, q \in Q \text{ such that } xpay \Rightarrow^{*}_{even} xqay \Rightarrow xo \Rightarrow t \text{ in } M.$$

Proof of Claim 6.9.D. Prove this claim by analogy with the proof of Claim 6.9.C.

Claim 6.9.E. For all $p \in Q$,

$$\langle p\curvearrowleft\rangle, \langle p\curvearrowright\rangle \in F' \text{ iff there is } f \in F \text{ such that } p \Rightarrow^{*}_{even} f \text{ in } M.$$

Proof of Claim 6.9.E. First, we establish the *only-if* part of this equivalence. By induction on the number of iterations of step (5) $i \geq 0$, we prove that $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in F'$ implies that there is $f \in F$ such that $q \Rightarrow^{*}_{even} f$ in M.

Basis. For $i = 0$, by the above construction technique, only $\langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle \in F'$ for all $f \in F$. Clearly, $f \Rightarrow^{0}_{even} f$ in M, so the basis holds true.

Induction Hypothesis. Assume that the implication holds for no more than j iterations of step (5), for some $j \in \mathbb{N}_0$.

Induction Step. Consider any $\langle p\curvearrowleft\rangle, \langle p\curvearrowright\rangle \in Q'$ belonging to F' from the $(j+1)$th iteration of step (5). Then, there exist $p \to q, p \to o \in R$, for some $o, q \in Q$, and $\langle o\curvearrowleft\rangle, \langle o\curvearrowright\rangle \in Q'$ that were added to F' during the jth iteration of step (5). By the induction hypothesis, there is $f \in F$ such that $o \Rightarrow^{*}_{even} f$ in M, so M can make

$$p \Rightarrow q \Rightarrow o \Rightarrow^{*}_{even} f.$$

Hence, by the definition of even computation, $p \Rightarrow^{*}_{even} f$ in M, which completes the induction step.

Now, we establish the *if* part of the equivalence stated in Claim 6.9.E. By induction on the number of moves $i \geq 0$, we show that $p \Rightarrow^{i}_{even} f$, where $f \in F$, in M implies $\langle p\curvearrowleft\rangle, \langle p\curvearrowright\rangle \in F'$.

Basis. Let $i = 0$, so $p \Rightarrow^{0}_{even} f$ in M. Then, $p = f$. Clearly, $\langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle \in F'$, as $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in F'$ for all $q \in F$. For $i = 1$, $p \Rightarrow^{1}_{even} f$ never occurs in M because, by Definition 6.1, every even computation is supposed to have an even number of moves. However, $p \Rightarrow^{1}_{even} f$ has one move. Therefore, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $p \Rightarrow^{j+2}_{even} f$ in M with $f \in F$. Let this computation start with the application of two consecutive rules of the forms

$$p \to q \text{ and } q \to o$$

from R, where $o, q \in Q$. Thus, we can express $p \Rightarrow^{j+2}_{even} f$ as

$$p \Rightarrow q \Rightarrow o \Rightarrow^{j}_{even} f$$

in M. Clearly, by the induction hypothesis, $\langle o\curvearrowleft\rangle, \langle o\curvearrowright\rangle \in F'$. Since $\langle o\curvearrowleft\rangle, \langle o\curvearrowright\rangle \in F'$ and $p \to q, q \to o \in R$, step (5) adds $\langle p\curvearrowleft\rangle$ and $\langle p\curvearrowright\rangle$ to F'. Thus, the induction step is completed, and Claim 6.9.E holds.

Based on Claims 6.9.A, 6.9.B, 6.9.C, 6.9.D, and 6.9.E, given above, we can conclude that for all $u, v \in \Sigma^*$ and $p, q \in Q$, $u\langle p\curvearrowleft\rangle v \Rightarrow^{*}_{even} \langle q\curvearrowleft\rangle$ or $u\langle p\curvearrowright\rangle v \Rightarrow^{*}_{even} \langle q\curvearrowright\rangle$ in M', where $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in F'$, iff there is $f \in F$ such that $upv \Rightarrow^{*}_{even} q \Rightarrow^{*}_{even} f$ in M. Considering this equivalence for $p = s$, $u\langle s\curvearrowleft\rangle v \Rightarrow^{*}_{even} \langle q\curvearrowleft\rangle$ or $u\langle s\curvearrowright\rangle v \Rightarrow^{*}_{even} \langle q\curvearrowright\rangle$ in M', where $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in F'$, iff there is $f \in F$ such that $usv \Rightarrow^{*}_{even} q \Rightarrow^{*}_{even} f$ in M. As follows from the construction technique, M' starts every computation from its initial state s', from which the same moves can be made as from the states $\langle s\curvearrowleft\rangle$ and $\langle s\curvearrowright\rangle$. In other words, M' starts each computation using either a rule of the form $as' \to t'$, for which $a\langle s\curvearrowleft\rangle \to t' \in R'$, or a rule of the form $s'a \to t'$, for which $\langle s\curvearrowright\rangle a \to t' \in R'$, where $a \in \Sigma$ and $t' \in Q'$. Consequently, $us'v \Rightarrow^{*}_{even} \langle q\curvearrowleft\rangle$ or $us'v \Rightarrow^{*}_{even} \langle q\curvearrowright\rangle$ in M', where $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in F'$, iff there is $f \in F$ such that $usv \Rightarrow^{*}_{even} q \Rightarrow^{*}_{even} f$ in M. Hence, $L(M')_{even} = L(M)_{even}$.

Obviously, $L(M')_{even} \subseteq L(M')$ follows directly from the definition of even computation. The opposite inclusion, $L(M') \subseteq L(M')_{even}$, follows directly from the construction technique above. Indeed, for each state of M' except s', according to the construction of R', all moves that lead to it read symbols in one direction, while all moves that can be performed from it read symbols in the opposite direction. From s', both left moves and right moves can be made, as no move ever leads to this state. Therefore, Lemma 6.9 holds. □

To illustrate the technique from the proof of Lemma 6.9, we provide the following example.

Example 6.10. Consider the IE2GFA

$$M = (\{s, q_1, q_2, f\}, \{a, b, c, d\}, R, s, \{f\}),$$

where R contains the following six rules (see Figure 6.2):

$$\begin{array}{lll} s \to q_1, & aaf \to q_2, & fdd \to q_2, \\ q_1 \to f, & q_2bb \to f, & ccq_2 \to f. \end{array}$$

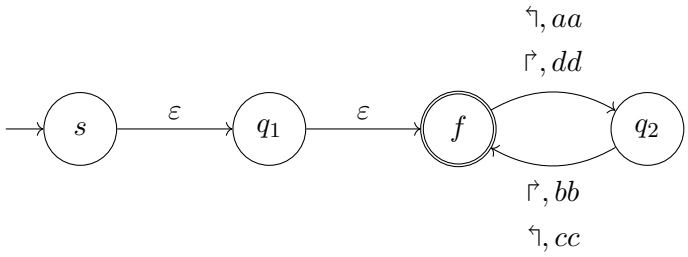

Figure 6.2: State diagram of the IE2GFA M from Example 6.10.

Clearly, $L(M)_{even} = \{a^{2n}b^{2n}, c^{2n}d^{2n} \mid n \geq 0\}$. Next, we apply the technique from the proof of Lemma 6.9 to M. However, since this technique introduces many unreachable states into the resulting automaton, we omit them from its definition for simplicity. This gives us the ε-free IE2SFA

$$M' = (Q', \{a, b, c, d\}, R', s', \{\langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle, s'\}),$$

with $Q' = \{s', \langle f\curvearrowleft\rangle, \langle afbb\curvearrowleft\rangle, \langle afb\curvearrowleft\rangle, \langle fb\curvearrowleft\rangle, \langle f\curvearrowright\rangle, \langle ccfd\curvearrowright\rangle, \langle cfd\curvearrowright\rangle, \langle cf\curvearrowright\rangle\}$ and R' consisting of the following rules (see Figure 6.3):

$$\begin{aligned}
&as' \to \langle afbb\curvearrowleft\rangle, && s'd \to \langle ccfd\curvearrowright\rangle,\\
&\langle afbb\curvearrowleft\rangle b \to \langle afb\curvearrowleft\rangle, && c\langle ccfd\curvearrowright\rangle \to \langle cfd\curvearrowright\rangle,\\
&a\langle afb\curvearrowleft\rangle \to \langle fb\curvearrowleft\rangle, && \langle cfd\curvearrowright\rangle d \to \langle cf\curvearrowright\rangle,\\
&\langle fb\curvearrowleft\rangle b \to \langle f\curvearrowleft\rangle, && c\langle cf\curvearrowright\rangle \to \langle f\curvearrowright\rangle,\\
&a\langle f\curvearrowleft\rangle \to \langle afbb\curvearrowleft\rangle, && \langle f\curvearrowright\rangle d \to \langle ccfd\curvearrowright\rangle.
\end{aligned}$$

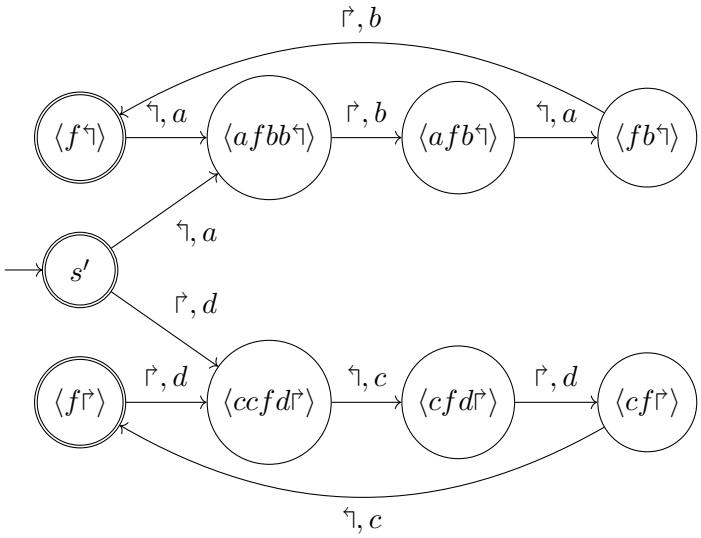

Figure 6.3: State diagram of the ε-free IE2SFA M' from Example 6.10.

On the string *ccccdddd*, which M accepts by the even computation

$$ccccsdddd \Rightarrow ccccq_1dddd \Rightarrow ccccfdddd \Rightarrow ccccq_2dd \Rightarrow ccfdd \Rightarrow ccq_2 \Rightarrow f,$$

M' performs the even computation

$$\begin{aligned} cccs'dddd \Rightarrow cccc\langle ccfd\curvearrowright\rangle ddd \Rightarrow ccc\langle cfd\curvearrowright\rangle ddd \Rightarrow ccc\langle cf\curvearrowright\rangle dd \Rightarrow cc\langle f\curvearrowright\rangle dd \\ \Rightarrow cc\langle ccfd\curvearrowright\rangle d \Rightarrow c\langle cfd\curvearrowright\rangle d \Rightarrow c\langle cf\curvearrowright\rangle \Rightarrow \langle f\curvearrowright\rangle. \end{aligned}$$

Observe that $L(M') = L(M')_{even} = L(M)_{even}$.

Theorem 6.11. ${}_{IE2GFA}^{\varepsilon}\Phi_{even} = {}_{IE2SFA}\Phi_{even}$.

Proof. ${}_{IE2SFA}\Phi_{even} \subseteq {}_{IE2GFA}^{\varepsilon}\Phi_{even}$ follows directly from the definition of an ε-free IE2SFA (see Definition 4.5). ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \subseteq {}_{IE2SFA}\Phi_{even}$ follows from Lemma 6.9, so this theorem holds. □

Lemma 6.12. For every IE2GFA M, there is an IE2GFA M' such that $L(M')_{init\text{-}even} = L(M)_{even}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. Without any loss of generality, assume that $s' \notin Q$. Clearly, the IE2GFA

$$M' = (Q \cup \{s'\}, \Sigma, R \cup \{s' \to s\}, s', F).$$

satisfies $L(M')_{init\text{-}even} = L(M)_{even}$, so Lemma 6.12 holds. □

Theorem 6.13. ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \subset {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even}$.

Proof. ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \subseteq {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even}$ follows directly from Lemma 6.12. Next, we prove that ${}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} \setminus {}_{IE2GFA}^{\varepsilon}\Phi_{even} \neq \varnothing$. Consider the language $K = \{a\}$. Clearly, the IE2GFA

$$M = (\{s, f\}, \{a\}, \{sa \to f\}, s, \{f\})$$

satisfies $L(M)_{init\text{-}even} = K$. However, by Theorem 6.7, there is no IE2GFA M' such that $L(M')_{even} = K$, so $K \in {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} \setminus {}_{IE2GFA}^{\varepsilon}\Phi_{even}$. Hence, ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \subset {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even}$. □

Lemma 6.14. For every IE2GFA $M = (Q, \Sigma, R, s, F)$, there exists an IE2SFA $M' = (Q', \Sigma, R', s', F')$ such that

(i) $r \in R'$ implies $\mathrm{rhs}(r) \neq s'$, $|\mathrm{lhs}(r)| = 1$ implies $\mathrm{lhs}(r) = s'$, and $s' \notin F'$;

(ii) $L(M') = L(M')_{init\text{-}even} = L(M)_{init\text{-}even}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. From M, we construct an IE2SFA $M' = (Q', \Sigma, R', s', F')$ satisfying the properties of Lemma 6.14. Let $\hat{M} = (\hat{Q}, \Sigma, \hat{R}, \hat{s}, \hat{F})$ be the ε-free IE2SFA constructed from M by the technique described in the proof of Lemma 6.9. Recall that $L(\hat{M}) = L(\hat{M})_{even} = L(M)_{even}$ and $\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \in \hat{Q}$ for all $q \in Q$. Introduce a new symbol s'—the start state of M'. Set $\bar{Q} = \{\langle xqy\rangle \mid q \in Q, x, y \in \Sigma^*, 1 \leq |x| + |y| \leq k - 1, \mathrm{abs}(|x| - |y|) \leq 1\}$, where $k = \max\{|\mathrm{lhs}(r)| \mid r \in R\}$. Without any loss of generality, assume that $\hat{Q} \cap \bar{Q} = \varnothing$ and $s' \notin \hat{Q} \cup \bar{Q}$. Set $Q' = (\hat{Q} \setminus \{\hat{s}\}) \cup \bar{Q} \cup \{s'\}$ and $F' = \hat{F} \setminus \{\hat{s}\}$. Initially, set $R' = \hat{R} \setminus \{a\hat{s} \to q, \hat{s}a \to q \mid q \in \hat{Q}, a \in \Sigma\}$. Then, extend R' in the following way:

(1) for each rule of the form $as \to q$ or $sa \to q$ from R, where $a \in \Sigma \cup \{\varepsilon\}$ and $q \in Q$, add both $s'a \to \langle q\curvearrowleft\rangle$ and $s'a \to \langle q\curvearrowright\rangle$ to R';

(2) for each rule of the form $a_1 \dots a_n a_{n+1} \dots a_{2n} s \to q$ or $s a_1 \dots a_n a_{n+1} \dots a_{2n} \to q$ from R, where $q \in Q$, $a_i \in \Sigma$, $1 \leq i \leq 2n$, and $n \geq 1$, extend R' by adding

$$
\begin{aligned}
s' &\to \langle a_1 \dots a_n q a_{n+1} \dots a_{2n} \rangle, \\
a_n \langle a_1 \dots a_n q a_{n+1} \dots a_{2n} \rangle &\to \langle a_1 \dots a_{n-1} q a_{n+1} \dots a_{2n} \rangle, \\
\langle a_1 \dots a_{n-1} q a_{n+1} \dots a_{2n} \rangle a_{n+1} &\to \langle a_1 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle, \\
a_{n-1} \langle a_1 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle &\to \langle a_1 \dots a_{n-2} q a_{n+2} \dots a_{2n} \rangle, \\
&\vdots \\
\langle q a_{2n} \rangle a_{2n} &\to \langle q^{\leftarrow\!\urcorner} \rangle, \\
\langle a_1 \dots a_n q a_{n+1} \dots a_{2n} \rangle a_{n+1} &\to \langle a_1 \dots a_n q a_{n+2} \dots a_{2n} \rangle, \\
a_n \langle a_1 \dots a_n q a_{n+2} \dots a_{2n} \rangle &\to \langle a_1 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle, \\
\langle a_1 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle a_{n+2} &\to \langle a_1 \dots a_{n-1} q a_{n+3} \dots a_{2n} \rangle, \\
&\vdots \\
a_1 \langle a_1 q \rangle &\to \langle q^{\urcorner\!\rightarrow} \rangle;
\end{aligned}
$$

(3) for each rule of the form $a_0 \dots a_n a_{n+1} \dots a_{2n} s \to q$ or $s a_0 \dots a_n a_{n+1} \dots a_{2n} \to q$ from R, where $q \in Q$, $a_i \in \Sigma$, $0 \leq i \leq 2n$, and $n \geq 1$, extend R' by adding

$$
\begin{aligned}
s' a_n &\to \langle a_0 \dots a_{n-1} q a_{n+1} \dots a_{2n} \rangle, \\
a_{n-1} \langle a_0 \dots a_{n-1} q a_{n+1} \dots a_{2n} \rangle &\to \langle a_0 \dots a_{n-2} q a_{n+1} \dots a_{2n} \rangle, \\
\langle a_0 \dots a_{n-2} q a_{n+1} \dots a_{2n} \rangle a_{n+1} &\to \langle a_0 \dots a_{n-2} q a_{n+2} \dots a_{2n} \rangle, \\
&\vdots \\
\langle q a_{2n} \rangle a_{2n} &\to \langle q^{\leftarrow\!\urcorner} \rangle, \\
\langle a_0 \dots a_{n-1} q a_{n+1} \dots a_{2n} \rangle a_{n+1} &\to \langle a_0 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle, \\
a_{n-1} \langle a_0 \dots a_{n-1} q a_{n+2} \dots a_{2n} \rangle &\to \langle a_0 \dots a_{n-2} q a_{n+2} \dots a_{2n} \rangle, \\
&\vdots \\
a_0 \langle a_0 q \rangle &\to \langle q^{\urcorner\!\rightarrow} \rangle.
\end{aligned}
$$

Basic Idea. M' simulates any initialized even computation in M by a sequence of moves, the first of which reads at most one symbol, while all the remaining moves read exactly one symbol at a time and can, in fact, always be made in such a way that they form an even computation. To explain step (1), simply assume that M performs the first move of an initialized even computation according to a rule of the form $as \to q$ or $sa \to q$, where $q \in Q$ and $a \in \Sigma$. Then, this step introduces $s'a \to \langle q^{\leftarrow\!\urcorner} \rangle$ and $s'a \to \langle q^{\urcorner\!\rightarrow} \rangle$ into R'. Clearly, by applying one of these rules, M' simulates the first move of the initialized even computation in M. Notice that both of the newly introduced rules, $s'a \to \langle q^{\leftarrow\!\urcorner} \rangle$ and $s'a \to \langle q^{\urcorner\!\rightarrow} \rangle$, are right because, by the definition of initialized even computation, there are no restrictions based on the direction of the first move of this computation. To explain step (2), assume that M performs the first move of an initialized even computation according to a rule of the form $a_1 \dots a_n a_{n+1} \dots a_{2n} s \to q$ or $s a_1 \dots a_n a_{n+1} \dots a_{2n} \to q$, where $q \in Q$ and $a_i \in \Sigma$,

$1 \leq i \leq 2n$, for some $n \geq 1$. Consider the sequence of rules introduced into R' in step (2). Observe that once M' applies its first rule, it has to continue by applying the rules from this sequence until it reaches either the state $\langle q^{\curvearrowleft} \rangle$ or $\langle q^{\curvearrowright} \rangle$. During this process, M' reads the string $a_1 \ldots a_n a_{n+1} \ldots a_{2n}$. Thus, the first move of the initialized even computation in M is simulated. Notice that the first rule, $s' \to \langle a_1 \ldots a_n q a_{n+1} \ldots a_{2n} \rangle$, is an ε-rule. This is because the sequence $a_1 \ldots a_n a_{n+1} \ldots a_{2n}$ contains an even number of symbols, but any initialized even computation always consists of an odd number of moves. Step (3) is analogous to step (2), except that the first rule of the introduced sequence is of the form $s' a_n \to \langle a_0 \ldots a_{n-1} q a_{n+1} \ldots a_{2n} \rangle$, where $q \in Q$ and $a_i \in \Sigma$, $0 \leq i \leq 2n$, $n \geq 1$, as $a_0 \ldots a_n a_{n+1} \ldots a_{2n}$ consists of an odd number of symbols. The rest of an initialized even computation in M, more precisely, its even part, is simulated by M' in the same way as by $\hat{M}$ (for details see the proof of Lemma 6.9).

Now, we establish $L(M')_{\mathit{init\text{-}even}} = L(M)_{\mathit{init\text{-}even}}$ formally. From the proof of Lemma 6.9, it follows that for all $p, q \in Q$ and $u, v \in \Sigma^*$, $u\langle q^{\curvearrowleft} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowleft} \rangle$ or $u\langle q^{\curvearrowright} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowright} \rangle$ in M', where $\langle p^{\curvearrowleft} \rangle, \langle p^{\curvearrowright} \rangle \in F'$, iff there is $f \in F$ such that $uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f$ in M. Then, according to steps (1), (2), and (3) of the construction technique of M', the following holds:

(i) $us'av \Rightarrow u\langle q^{\curvearrowleft} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowleft} \rangle$ or $us'av \Rightarrow u\langle q^{\curvearrowright} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowright} \rangle$ in M', where $\langle p^{\curvearrowleft} \rangle, \langle p^{\curvearrowright} \rangle \in F'$, iff there is $f \in F$ such that $uasv \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f$ or $usav \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f$ in M, where $a \in \Sigma \cup \{\varepsilon\}$;

(ii) for all $n \geq 1$,

$$\begin{aligned} ua_1 \ldots a_n s' a_{n+1} \ldots a_{2n} v &\Rightarrow ua_1 \ldots a_n \langle a_1 \ldots a_n q a_{n+1} \ldots a_{2n} \rangle a_{n+1} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_{n-1} \langle a_1 \ldots a_{n-1} q a_{n+1} \ldots a_{2n} \rangle a_{n+1} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_{n-1} \langle a_1 \ldots a_{n-1} q a_{n+2} \ldots a_{2n} \rangle a_{n+2} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_{n-2} \langle a_1 \ldots a_{n-2} q a_{n+2} \ldots a_{2n} \rangle a_{n+2} \ldots a_{2n} v \\ &\vdots \\ &\Rightarrow u\langle q a_{2n} \rangle a_{2n} v \Rightarrow u\langle q^{\curvearrowleft} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowleft} \rangle \end{aligned}$$

or

$$\begin{aligned} ua_1 \ldots a_n s' a_{n+1} \ldots a_{2n} v &\Rightarrow ua_1 \ldots a_n \langle a_1 \ldots a_n q a_{n+1} \ldots a_{2n} \rangle a_{n+1} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_n \langle a_1 \ldots a_n q a_{n+2} \ldots a_{2n} \rangle a_{n+2} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_{n-1} \langle a_1 \ldots a_{n-1} q a_{n+2} \ldots a_{2n} \rangle a_{n+2} \ldots a_{2n} v \\ &\Rightarrow ua_1 \ldots a_{n-1} \langle a_1 \ldots a_{n-1} q a_{n+3} \ldots a_{2n} \rangle a_{n+3} \ldots a_{2n} v \\ &\vdots \\ &\Rightarrow ua_1 \langle a_1 q \rangle v \Rightarrow u\langle q^{\curvearrowright} \rangle v \Rightarrow^*_{even} \langle p^{\curvearrowright} \rangle \end{aligned}$$

in M', where $\langle p^{\curvearrowleft} \rangle, \langle p^{\curvearrowright} \rangle \in F'$, iff there is $f \in F$ such that

$$ua_1 \ldots a_{2n} sv \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f \text{ or } usa_1 \ldots a_{2n} v \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f$$

in M, where $a_i \in \Sigma$, $1 \leq i \leq 2n$;

(iii) for all $n \geq 1$,

$$\begin{aligned} ua_0 \ldots a_{n-1} s' a_n \ldots a_{2n} v &\Rightarrow ua_0 \ldots a_{n-1} \langle a_0 \ldots a_{n-1} q a_{n+1} \ldots a_{2n} \rangle a_{n+1} \ldots a_{2n} v \\ &\Rightarrow ua_0 \ldots a_{n-2} \langle a_0 \ldots a_{n-2} q a_{n+1} \ldots a_{2n} \rangle a_{n+1} \ldots a_{2n} v \end{aligned}$$

$$\Rightarrow ua_0 \dots a_{n-2}\langle a_0 \dots a_{n-2} q a_{n+2} \dots a_{2n}\rangle a_{n+2} \dots a_{2n} v$$
$$\vdots$$
$$\Rightarrow u\langle q a_{2n}\rangle a_{2n} v \Rightarrow u\langle q{\curvearrowleft}\rangle v \Rightarrow^*_{even} \langle p{\curvearrowleft}\rangle$$

or

$$\begin{aligned} ua_0 \dots a_{n-1} s' a_n \dots a_{2n} v &\Rightarrow ua_1 \dots a_{n-1}\langle a_0 \dots a_{n-1} q a_{n+1} \dots a_{2n}\rangle a_{n+1} \dots a_{2n} v \\ &\Rightarrow ua_1 \dots a_{n-1}\langle a_0 \dots a_{n-1} q a_{n+2} \dots a_{2n}\rangle a_{n+2} \dots a_{2n} v \\ &\Rightarrow ua_1 \dots a_{n-2}\langle a_0 \dots a_{n-2} q a_{n+2} \dots a_{2n}\rangle a_{n+2} \dots a_{2n} v \\ &\vdots \\ &\Rightarrow ua_0\langle a_0 q\rangle v \Rightarrow u\langle q{\curvearrowright}\rangle v \Rightarrow^*_{even} \langle p{\curvearrowright}\rangle \end{aligned}$$

in M', where $\langle p{\curvearrowleft}\rangle, \langle p{\curvearrowright}\rangle \in F'$, iff there is $f \in F$ such that

$$ua_0 \dots a_{2n} sv \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f \text{ or } usa_0 \dots a_{2n} v \Rightarrow uqv \Rightarrow^*_{even} p \Rightarrow^*_{even} f$$

in M, where $a_i \in \Sigma$, $0 \leq i \leq 2n$.

Clearly, from states of the forms $\langle q{\curvearrowleft}\rangle$ and $\langle q{\curvearrowright}\rangle$, where $q \in Q$, M' can only make left moves and right moves, respectively. Thus, by the definition of initialized even computation, we can express the previous equivalences as follows:

(i) $us'av \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowleft}\rangle$ or $us'av \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowright}\rangle$ in M', where $\langle p{\curvearrowleft}\rangle, \langle p{\curvearrowright}\rangle \in F'$, iff there is $f \in F$ such that $uasv \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f$ or $usav \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f$ in M, where $a \in \Sigma \cup \{\varepsilon\}$;

(ii) for all $n \geq 1$,

$$ua_1 \dots a_n s' a_{n+1} \dots a_{2n} v \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowleft}\rangle \text{ or } ua_1 \dots a_n s' a_{n+1} \dots a_{2n} v \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowright}\rangle$$

in M', where $\langle p{\curvearrowleft}\rangle, \langle p{\curvearrowright}\rangle \in F'$, iff there is $f \in F$ such that

$$ua_1 \dots a_{2n} sv \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f \text{ or } usa_1 \dots a_{2n} v \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f$$

in M, where $a_i \in \Sigma$, $1 \leq i \leq 2n$;

(iii) for all $n \geq 1$,

$$ua_0 \dots a_{n-1} s' a_n \dots a_{2n} v \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowleft}\rangle \text{ or } ua_0 \dots a_{n-1} s' a_n \dots a_{2n} v \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowright}\rangle$$

in M', where $\langle p{\curvearrowleft}\rangle, \langle p{\curvearrowright}\rangle \in F'$, iff there is $f \in F$ such that

$$ua_0 \dots a_{2n} sv \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f \text{ or } usa_0 \dots a_{2n} v \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f$$

in M, where $a_i \in \Sigma$, $0 \leq i \leq 2n$.

Based on the above information, we can safely conclude that for all $w_1, w_2 \in \Sigma^*$ and $p \in Q$, $w_1 s' w_2 \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowleft}\rangle$ or $w_1 s' w_2 \Rightarrow^*_{init\text{-}even} \langle p{\curvearrowright}\rangle$, where $\langle p{\curvearrowleft}\rangle, \langle p{\curvearrowright}\rangle \in F'$, iff there is $f \in F$ such that $w_1 s w_2 \Rightarrow^*_{init\text{-}even} p \Rightarrow^*_{even} f$ in M. Hence, $L(M')_{init\text{-}even} = L(M)_{init\text{-}even}$.

Obviously, $L(M')_{init\text{-}even} \subseteq L(M')$. Observe, however, that by the construction of M', there is no $w \in \Sigma^*$ such that $w \in L(M') \setminus L(M')_{init\text{-}even}$, so also $L(M') \subseteq L(M')_{init\text{-}even}$. In addition, notice that s' can never occur on the right side of any rule, that every ε-rule always has s' on its left-hand side, and that s' can never be a final state. Therefore, Lemma 6.12 holds. □

The next example demonstrates the construction technique described in the previous proof of Lemma 6.14.

Example 6.15. Consider the ε-free IE2GFA

$$M = (\{s, q, f\}, \{a, b, c\}, R, s, \{f\})$$

with the following five rules in R (see Figure 6.4):

$$\begin{aligned} as &\to f, & fb &\to q, \\ scc &\to f, & bq &\to f, \\ cbcs &\to f. \end{aligned}$$

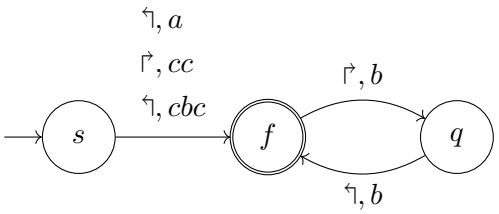

Figure 6.4: State diagram of the ε-free IE2GFA M from Example 6.15.

Now, we apply the technique from the proof of Lemma 6.14 to M. Note that, as in Example 6.10, we do not list the unreachable states of the resulting automaton for simplicity. Thus, we obtain the IE2SFA

$$M' = (\{s', \langle cfc\rangle, \langle fc\rangle, \langle cf\rangle, \langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle, \langle bf\curvearrowright\rangle\}, \{a, b, c\}, R', s', \{\langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle\})$$

with R' containing the following rules (see Figure 6.5):

$$\begin{aligned} &s'a \to \langle f\curvearrowleft\rangle, && s' \to \langle cfc\rangle, && c\langle cfc\rangle \to \langle fc\rangle, && c\langle cf\rangle \to \langle f\curvearrowright\rangle, && \langle f\curvearrowright\rangle b \to \langle bf\curvearrowright\rangle, \\ &s'a \to \langle f\curvearrowright\rangle, && s'b \to \langle cfc\rangle, && \langle cfc\rangle c \to \langle cf\rangle, && \langle fc\rangle c \to \langle f\curvearrowleft\rangle, && b\langle bf\curvearrowright\rangle \to \langle f\curvearrowright\rangle. \end{aligned}$$

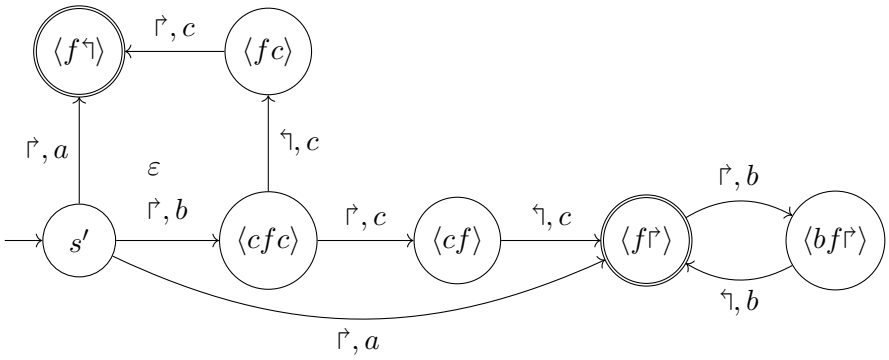

Figure 6.5: State diagram of the IE2SFA M' from Example 6.15.

Consider the string *bbcbcbb*, which M accepts by the initialized even computation

$$bbcbcsbb \Rightarrow bbfbb \Rightarrow bbqb \Rightarrow bfb \Rightarrow bq \Rightarrow f.$$

M' accepts this string by the initialized even computation

$$bbcs'bcbb \Rightarrow bbc\langle cfc\rangle cbb \Rightarrow bbc\langle cf\rangle bb \Rightarrow bb\langle f\curvearrowright\rangle bb \Rightarrow bb\langle bf\curvearrowright\rangle b \Rightarrow b\langle f\curvearrowright\rangle b \Rightarrow b\langle bf\curvearrowright\rangle \Rightarrow \langle f\curvearrowright\rangle.$$

Observe that $L(M') = L(M')_{init\text{-}even} = L(M)_{init\text{-}even} = \{b^n x b^n \mid x \in \{a, cc, cbc\}, n \geq 0\}$.

Theorem 6.16. ${}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} = {}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even}$.

Proof. The inclusion ${}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even} \subseteq {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even}$ is obvious. The opposite inclusion, ${}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} \subseteq {}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even}$, follows from Lemma 6.14. □

Theorem 6.17. ${}_{IE2SFA}\Phi_{init\text{-}even} \subset {}_{odd}\Phi$.

Proof. According to Definition 6.1, each initialized even computation consists of an odd number of moves. Therefore, no ε-free IE2SFA can ever accept any even-length string in this way, since it always reads exactly one symbol per move. Consequently, each language in ${}_{IE2SFA}\Phi_{init\text{-}even}$ consists of odd-length strings only, so ${}_{IE2SFA}\Phi_{init\text{-}even} \subset {}_{odd}\Phi$. □

From Theorems 6.7 and 6.17, we obtain the following corollary.

Corollary 6.18. ${}_{IE2GFA}^{\varepsilon}\Phi_{even} \cap {}_{IE2SFA}\Phi_{init\text{-}even} = \varnothing$.

Theorem 6.19. ${}_{IE2SFA}\Phi_{init\text{-}even} \subset {}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even}$.

Proof. Clearly, ${}_{IE2SFA}\Phi_{init\text{-}even} \subseteq {}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even}$. Next, we show that this inclusion is proper. Consider the language $K = \{\varepsilon\}$. Clearly, $K \in {}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even}$. However, as follows from Theorem 6.17, there is no ε-free IE2SFA M satisfying $L(M)_{init\text{-}even} = K$. Therefore, ${}_{IE2SFA}^{\varepsilon}\Phi_{init\text{-}even} \setminus {}_{IE2SFA}\Phi_{init\text{-}even} \neq \varnothing$, and Theorem 6.19 holds. □

Theorem 6.20. ${}_{IE2SFA}\Phi_{init\text{-}even}$ is incomparable with any of these families of languages—${}_{sing}\Phi$, ${}_{fin}\Phi$, and ${}_{reg}\Phi$.

Proof. Let $L \in {}_{IE2SFA}\Phi_{init\text{-}even}$. By Theorem 6.17, $x \in L$ implies that $|x|$ is odd, so $\{aa\} \notin {}_{IE2SFA}\Phi_{init\text{-}even}$. However, $\{aa\} \in {}_{sing}\Phi$. Clearly, $\{a\} \in {}_{IE2SFA}\Phi_{init\text{-}even} \cap {}_{sing}\Phi$ and $\{a^n bc^n \mid n \geq 0\} \in {}_{IE2SFA}\Phi_{init\text{-}even} \setminus {}_{sing}\Phi$. Thus, ${}_{IE2SFA}\Phi_{init\text{-}even}$ and ${}_{sing}\Phi$ are incomparable. The rest of this proof proceeds analogously. □

Lemma 6.21. For every IE2GFA M, there is an ε-free IE2SFA M' such that $L(M') = L(M')_{alt} = L(M)_{init\text{-}even}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA, and let $\hat{M} = (\hat{Q}, \Sigma, \hat{R}, \hat{s}, \hat{F})$ be the IE2SFA constructed from M by the technique from the proof of Lemma 6.14. Recall that $L(\hat{M}) = L(\hat{M})_{init\text{-}even} = L(M)_{init\text{-}even}$, $r \in \hat{R}$ implies $\mathrm{rhs}(r) \neq \hat{s}$, $|\mathrm{lhs}(r)| = 1$ implies $\mathrm{lhs}(r) = \hat{s}$, and $\hat{s} \notin \hat{F}$. Next, we construct the ε-free IE2SFA

$$M' = ((\hat{Q} \setminus \{\hat{s}\}) \cup \{s'\}, \Sigma, R', s', F'),$$

where $s' \notin \hat{Q}$,

$$\begin{aligned} R' = (&\hat{R} \setminus \{\hat{s}a \to q \mid, q \in \hat{Q}, a \in \Sigma \cup \{\varepsilon\}\}) \\ &\cup \{as' \to q, s'a \to q \mid \hat{s}a \to q \in \hat{R}, q \in \hat{Q}, a \in \Sigma\} \\ &\cup \{as' \to q \mid \hat{s} \to p, ap \to q \in \hat{R}, p, q \in \hat{Q}, a \in \Sigma\} \\ &\cup \{s'a \to q \mid \hat{s} \to p, pa \to q \in \hat{R}, p, q \in \hat{Q}, a \in \Sigma\} \end{aligned}$$

and

$$F' = \begin{cases} \hat{F} \cup \{s'\} & \text{if } \{f \in \hat{F} \mid \hat{s} \to f \in \hat{R}\} \neq \varnothing, \\ \hat{F} & \text{otherwise.} \end{cases}$$

Note that the construction of $\hat{M}$ implies that there are no rules of the form $a\hat{s} \to q$ in $\hat{R}$, where $q \in \hat{Q}$ and $a \in \Sigma$, so we do not need to consider them in the construction of R'. Observe that $L(M') = L(M')_{alt} = L(M)_{init\text{-}even}$. Therefore, Lemma 6.21 holds. □

Lemma 6.22. For each ε-free IE2SFA M, there is an IE2SFA M' such that $L(M')_{init\text{-}even} = L(M)_{alt}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $\hat{Q} = \{\langle q\curvearrowleft\rangle, \langle q\curvearrowright\rangle \mid q \in Q\}$. Without any loss of generality, assume that $Q \cap \hat{Q} = \varnothing$ and $s' \notin Q \cup \hat{Q}$. Now, construct the IE2SFA

$$M' = (Q \cup \hat{Q} \cup \{s'\}, \Sigma, R', s', F'),$$

where

$$\begin{aligned} R' = R &\cup \{s'a \to \langle p\curvearrowright\rangle \mid as \to p \in R, p \in Q, a \in \Sigma\} \\ &\cup \{s'a \to \langle p\curvearrowleft\rangle \mid sa \to p \in R, p \in Q, a \in \Sigma\} \\ &\cup \{\langle p\curvearrowright\rangle a \to q \mid pa \to q \in R, p, q \in Q, a \in \Sigma\} \\ &\cup \{a\langle p\curvearrowleft\rangle \to q \mid ap \to q \in R, p, q \in Q, a \in \Sigma\} \\ &\cup \{s' \to s\} \end{aligned}$$

and $F' = F \cup \{\langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle \mid f \in F\}$. Observe that $L(M')_{init\text{-}even} = L(M)_{alt}$. Hence, Lemma 6.22 holds. □

Theorem 6.23. ${}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} = {}_{IE2SFA}\Phi_{alt}$.

Proof. ${}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even} \subseteq {}_{IE2SFA}\Phi_{alt}$ follows from Lemma 6.21. The opposite inclusion, ${}_{IE2SFA}\Phi_{alt} \subseteq {}_{IE2GFA}^{\varepsilon}\Phi_{init\text{-}even}$, follows from Lemma 6.22. □

6.3 Equivalence with Even Linear Grammars

This section demonstrates that the computational power of IE2GFAs working under initialized even computation is the same as the generative power of ELGs.

Lemma 6.24. For every IE2GFA M, there is an ELG G such that $L(G) = L(M)_{init\text{-}even}$.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an IE2GFA. From M, we next construct an ELG $G = (N, T, P, S)$ such that $L(G) = L(M)_{init\text{-}even}$. Introduce a new symbol S—the start nonterminal of G. Set $N' = \{\langle qd\rangle \mid q \in Q, d \in \{\curvearrowleft, \curvearrowright\}\}$. Without any loss of generality, assume that $S \notin N'$. Set $N = N' \cup \{S\}$ and $T = \Sigma$. P is then constructed as follows:

(1) for each $f \in F$, add $S \to \langle f\curvearrowleft\rangle$ and $S \to \langle f\curvearrowright\rangle$ to P;

(2) for each rule of the form $xs \to q$ or $sx \to q$ from R, where $q \in Q$ and $x \in \Sigma^*$, add $\langle q\curvearrowleft\rangle \to x$ and $\langle q\curvearrowright\rangle \to x$ to P;

(3) for each $xq \to p, py \to o \in R$, where $o, p, q \in Q$, $x, y \in \Sigma^*$, and $|x| = |y|$, add $\langle o\curvearrowleft\rangle \to x\langle q\curvearrowleft\rangle y$ to P;

(4) for each $qy \to p, xp \to o \in R$, where $o, p, q \in Q$, $x, y \in \Sigma^*$, and $|x| = |y|$, add $\langle o\curvearrowright\rangle \to x\langle q\curvearrowright\rangle y$ to P;

Basic Idea. G simulates any initialized even computation of M in reverse. It starts by generating a nonterminal of the form $\langle f\curvearrowleft\rangle$ or $\langle f\curvearrowright\rangle$ with $f \in F$, which corresponds to a final state (see step (1)). After this initial derivation step, G simulates every two-move even computation made by M according to two consecutive rules of the forms $xq \to p$ and $py \to o$, where $o, p, q \in Q$ and $x, y \in \Sigma^*$, by using a rule of the form $\langle o\curvearrowleft\rangle \to x\langle q\curvearrowleft\rangle y$ (see step (3)). Notice that the first rule, $xq \to p$, is a left rule. Step (4) is analogous to step (3), except that the first of the two consecutive rules is a right rule. As can be seen, if the even part of an initialized even computation in M starts with a left rule, it is simulated in G by a derivation over nonterminals of the form $\langle q\curvearrowleft\rangle$, where $q \in Q$; otherwise, it is simulated by a derivation over nonterminals of the form $\langle q\curvearrowright\rangle$ with $q \in Q$. The simulation process is completed by applying a rule of the form $\langle q\curvearrowleft\rangle \to x$ or $\langle q\curvearrowright\rangle \to x$, where $q \in Q$ and $x \in \Sigma^*$. Thus, the symbol sequence read by the first move of an initialized even computation in M is generated, and a string of terminals in G is obtained (see step (2)).

Let us now establish $L(G) = L(M)_{init\text{-}even}$ formally. We start by proving the following two claims.

Claim 6.24.A. For all $u, v \in \Sigma^*$ and $p, q \in Q$,

$$\langle q\curvearrowleft\rangle \Rightarrow^* u\langle p\curvearrowleft\rangle v \text{ in } G \text{ iff } upv \Rightarrow^*_{even} q \text{ in } M,$$

where $upv \Rightarrow^*_{even} q$ starts with a left move (unless it is an empty sequence of moves).

Proof of Claim 6.24.A. First, we establish the *only-if* part of this equivalence. By induction on the number of derivation steps $i \geq 0$, we prove that $\langle q\curvearrowleft\rangle \Rightarrow^i u\langle p\curvearrowleft\rangle v$ in G implies that there is $upv \Rightarrow^*_{even} q$ in M that starts with a left move (or consists of no moves).

Basis. Let $i = 0$, so $\langle q\curvearrowleft\rangle \Rightarrow^0 u\langle p\curvearrowleft\rangle v$ in G. Then, $q = p$ and $uv = \varepsilon$. Since $q \Rightarrow^0_{even} q$ in M, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all derivations consisting of no more than j steps, for some $j \in \mathbb{N}_0$.

Induction Step. Consider any derivation of the form $\langle q\curvearrowleft\rangle \Rightarrow^{j+1} u\langle p\curvearrowleft\rangle v$ in G. Let this derivation start with the application of a rule of the form

$$\langle q\curvearrowleft\rangle \to x\langle o\curvearrowleft\rangle y$$

from P, where $o \in Q$, $x, y \in \Sigma^*$, and $|x| = |y|$. Thus, we can express $\langle q\curvearrowleft\rangle \Rightarrow^{j+1} u\langle p\curvearrowleft\rangle v$ as

$$\langle q\curvearrowleft\rangle \Rightarrow x\langle o\curvearrowleft\rangle y \Rightarrow^j xu'\langle p\curvearrowleft\rangle v'y$$

in G, where $xu' = u$ and $v'y = v$. By the induction hypothesis, $u'pv' \Rightarrow^*_{even} o$ in M, and this computation starts with a left move (or consists of no moves at all). Step (3) constructs $\langle q\curvearrowleft\rangle \to x\langle o\curvearrowleft\rangle y \in P$ from two consecutive rules $xo \to t, ty \to q \in R$, for some $t \in Q$, so

$$xu'pv'y \Rightarrow^*_{even} xoy \Rightarrow ty \Rightarrow q$$

in M. Since $xu' = u$, $v'y = v$, and $|x| = |y|$, taking into account the properties of $u'pv' \Rightarrow^*_{even} o$, it follows that $upv \Rightarrow^*_{even} q$ in M. As we can see, $upv \Rightarrow^*_{even} q$ starts with a left move. Thus, the induction step is completed.

Next, we establish the *if* part of the equivalence stated in Claim 6.24.A. By induction on the number of moves $i \geq 0$, we prove that if there is $upv \Rightarrow^{i}_{even} q$ in M that starts with a left move (or consists of no moves), then $\langle q\curvearrowleft\rangle \Rightarrow^{*} u\langle p\curvearrowleft\rangle v$ in G.

Basis. For $i = 0$, $upv \Rightarrow^{0}_{even} q$ occurs in M only for $p = q$ and $uv = \varepsilon$. Clearly, $\langle q\curvearrowleft\rangle \Rightarrow^{0} \langle q\curvearrowleft\rangle$ in G. For $i = 1$, $upv \Rightarrow^{1}_{even} q$ never occurs in M, since, by Definition 6.1, every even computation is supposed to have an even number of moves; however, $upv \Rightarrow^{1}_{even} q$ has one move. Thus, the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves, for some $j \in \mathbb{N}_0$.

Induction Step. Let $upv \Rightarrow^{j+2}_{even} q$ in M, and let this computation end with the application of two consecutive rules of the forms

$$xo \to t \text{ and } ty \to q$$

from R, where $o, t \in Q$, $x, y \in \Sigma^{*}$, and $|x| = |y|$. Express $upv \Rightarrow^{j+2}_{even} q$ as

$$xu'pv'y \Rightarrow^{j}_{even} xoy \Rightarrow ty \Rightarrow q$$

in M, where $xu' = u$ and $v'y = v$. Observe that $u'pv' \Rightarrow^{j}_{even} o$ starts with a left move (or consists of no moves at all). Thus, by the induction hypothesis, $\langle o\curvearrowleft\rangle \Rightarrow^{*} u'\langle p\curvearrowleft\rangle v'$ in G. From $xo \to t, ty \to q \in R$, step (3) constructs $\langle q\curvearrowleft\rangle \to x\langle o\curvearrowleft\rangle y \in P$, so G can make

$$\langle q\curvearrowleft\rangle \Rightarrow x\langle o\curvearrowleft\rangle y \Rightarrow^{*} xu'\langle p\curvearrowleft\rangle v'y.$$

Because $xu' = u$ and $v'y = v$, it follows that $\langle q\curvearrowleft\rangle \Rightarrow^{*} u\langle p\curvearrowleft\rangle v$ in G. Thus, the induction step is completed, and Claim 6.24.A holds.

Claim 6.24.B. For all $u, v \in \Sigma^{*}$ and $p, q \in Q$,

$$\langle q\curvearrowright\rangle \Rightarrow^{*} u\langle p\curvearrowright\rangle v \text{ in } G \text{ iff } upv \Rightarrow^{*}_{even} q \text{ in } M,$$

where $upv \Rightarrow^{*}_{even} q$ starts with a right move (unless it is an empty sequence of moves).

Proof of Claim 6.24.B. This can be proved analogously with the proof of Claim 6.24.A.

As a consequence of Claims 6.24.A and 6.24.B, for all $u, v \in \Sigma^{*}$ and $p, q \in Q$, we have $\langle q\curvearrowleft\rangle \Rightarrow^{*} u\langle p\curvearrowleft\rangle v$ or $\langle q\curvearrowright\rangle \Rightarrow^{*} u\langle p\curvearrowright\rangle v$ in G iff $upv \Rightarrow^{*}_{even} q$ in M. As follows from the above construction technique, G starts every derivation by applying a rule of the form $S \to \langle fd\rangle$, where $f \in F$ and $d \in \{\curvearrowleft, \curvearrowright\}$, and ends it by applying a rule of the form $\langle pd\rangle \to x$, where $p \in Q$, $x \in \Sigma^{*}$, and $d \in \{\curvearrowleft, \curvearrowright\}$, constructed from $xs \to p \in R$ or $sx \to p \in R$ by step (2). Consequently, $S \Rightarrow \langle f\curvearrowleft\rangle \Rightarrow^{*} u\langle p\curvearrowleft\rangle v \Rightarrow uxv$ or $S \Rightarrow \langle f\curvearrowright\rangle \Rightarrow^{*} u\langle p\curvearrowright\rangle v \Rightarrow uxv$ in G iff $uxsv \Rightarrow upv \Rightarrow^{*}_{even} f$ or $usxv \Rightarrow upv \Rightarrow^{*}_{even} f$ in M. Hence, by the definition of initialized even computation, $S \Rightarrow \langle f\curvearrowleft\rangle \Rightarrow^{*} u\langle p\curvearrowleft\rangle v \Rightarrow uxv$ or $S \Rightarrow \langle f\curvearrowright\rangle \Rightarrow^{*} u\langle p\curvearrowright\rangle v \Rightarrow uxv$ in G iff $uxsv \Rightarrow^{*}_{init\text{-}even} f$ or $usxv \Rightarrow^{*}_{init\text{-}even} f$ in M. As a result, $L(G) = L(M)_{init\text{-}even}$, so Lemma 6.24 holds. □

The technique used in the previous proof of Lemma 6.24 is illustrated in the following example.

Example 6.25. Consider the IE2SFA

$$M = (\{s, q_1, q_2, f\}, \{a, b\}, R, s, \{f\}),$$

where R consists of the following rules (see Figure 6.6):

$s \to q_1$,	$aq_1 \to q_1$,	$q_1 \to q_2$,	$af \to f$,
$sa \to q_1$,	$q_1a \to q_1$,	$q_2 \to f$,	$fb \to f$.

Observe that $L(M)_{\textit{init-even}} = \{a^m b^n \mid 0 \leq n \leq m\}$. Indeed, under initialized even computation, M first moves from s to q_1, reading ε or a. Then, it reads an arbitrary even number of as, the same number in each of the two directions. Finally, M reads an equal number of as to the left and bs to the right.

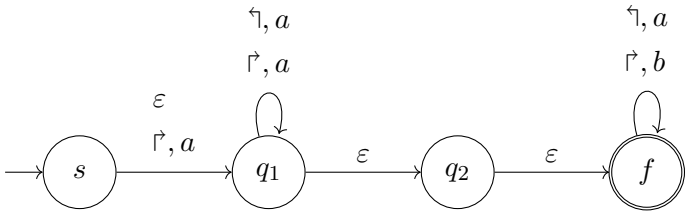

Figure 6.6: State diagram of the IE2SFA M from Example 6.25.

From M, the construction technique from the proof of Lemma 6.24 produces the ELG

$$G = (\{S, \langle f\curvearrowleft\rangle, \langle f\curvearrowright\rangle, \langle q_1\curvearrowleft\rangle, \langle q_1\curvearrowright\rangle, \langle q_2\curvearrowleft\rangle, \langle q_2\curvearrowright\rangle, \langle s\curvearrowleft\rangle, \langle s\curvearrowright\rangle\}, \{a, b\}, P, S)$$

with the following rules in P:

$S \to \langle f\curvearrowleft\rangle$,	$\langle f\curvearrowleft\rangle \to \langle q_1\curvearrowleft\rangle$,	$\langle q_1\curvearrowleft\rangle \to \varepsilon$,	$\langle q_2\curvearrowleft\rangle \to \langle s\curvearrowleft\rangle$,
$S \to \langle f\curvearrowright\rangle$,	$\langle f\curvearrowright\rangle \to \langle q_1\curvearrowright\rangle$,	$\langle q_1\curvearrowright\rangle \to \varepsilon$,	$\langle q_2\curvearrowright\rangle \to \langle s\curvearrowright\rangle$,
$\langle f\curvearrowleft\rangle \to a\langle f\curvearrowleft\rangle b$,	$\langle q_1\curvearrowleft\rangle \to a\langle q_1\curvearrowleft\rangle a$,	$\langle q_1\curvearrowleft\rangle \to a$,	$\langle q_1\curvearrowright\rangle \to a\langle s\curvearrowright\rangle a$,
$\langle f\curvearrowright\rangle \to a\langle f\curvearrowright\rangle b$,	$\langle q_1\curvearrowright\rangle \to a\langle q_1\curvearrowright\rangle a$,	$\langle q_1\curvearrowright\rangle \to a$.	

As we can see, G starts each derivation by rewriting S to $\langle fd\rangle$, for some $d \in \{\curvearrowleft, \curvearrowright\}$. After this initial derivation step, G continues by generating an equal number of as to the left and bs to the right by repeatedly replacing $\langle fd\rangle$ with $a\langle fd\rangle b$. Then, it rewrites $\langle fd\rangle$ to $\langle q_1 d\rangle$ and generates any even number of as by repeatedly replacing $\langle q_1 d\rangle$ with $a\langle q_1 d\rangle a$. Finally, G rewrites $\langle q_1 d\rangle$ to either ε or a. Clearly, $L(G) = L(M)_{\textit{init-even}}$.

For instance, for the initialized even computation

$$aaasaabb \Rightarrow aaaq_1abb \Rightarrow aaaq_1bb \Rightarrow aaq_1bb \Rightarrow aaq_2bb \Rightarrow aafbb \Rightarrow aafb \Rightarrow afb \Rightarrow af \Rightarrow f$$

in M accepting the string $aaaaabb$, the corresponding derivation in G is

$$S \Rightarrow \langle f\curvearrowright\rangle \Rightarrow a\langle f\curvearrowright\rangle b \Rightarrow aa\langle f\curvearrowright\rangle bb \Rightarrow aa\langle q_1\curvearrowright\rangle bb \Rightarrow aaa\langle q_1\curvearrowright\rangle abb \Rightarrow aaaaabb.$$

Lemma 6.26. For every ELG G, there is an IE2GFA M such that $L(M)_{\textit{init-even}} = L(G)$.

Proof. Let $G = (N, T, P, S)$ be an ELG and $M = (Q, \Sigma, R, s, F)$ be an IE2GFA constructed from G using the technique described in the proof of Lemma 5.3. Recall that $F = \{S\}$. As follows from the proof of Lemma 5.3, $uszv \Rightarrow uCv \Rightarrow^* S$ in M iff $S \Rightarrow^* uCv \Rightarrow uzv$ in G for all $C \in N$ and $u, v, z \in T^*$. According to the technique used for the construction of M, for each $A \to xBy \in P$, where $A, B \in N$ and $x, y \in T^*$, there are two consecutive rules $xB \to \langle A \to xBy\rangle, \langle A \to xBy\rangle y \to A \in R$, which are always applied one immediately after the other in the given order. Thus, $uCv \Rightarrow^* S$ consists of an even number of moves and is alternating, so $uCv \Rightarrow^*_{alt} S$. Furthermore, since G is even, $|x| = |y|$ always holds; hence, $uCv \Rightarrow^*_{alt} S$ is an even computation, so $uCv \Rightarrow^*_{even} S$. Consequently, $uszv \Rightarrow uCv \Rightarrow^*_{even} S$ in M iff $S \Rightarrow^* uCv \Rightarrow uzv$ in G. Hence, by the definition of initialized even computation, $uszv \Rightarrow^*_{init\text{-}even} S$ in M iff $S \Rightarrow^* uCv \Rightarrow uzv$ in G, so $L(M)_{init\text{-}even} = L(G)$. Therefore, Lemma 6.26 holds. □

Theorem 6.27. ${}_{IE2GFA}\overset{\varepsilon}{\Phi}_{init\text{-}even} = {}_{ELG}\overset{\varepsilon}{\Phi}$.

Proof. The inclusion ${}_{IE2GFA}\overset{\varepsilon}{\Phi}_{init\text{-}even} \subseteq {}_{ELG}\overset{\varepsilon}{\Phi}$ follows from Lemma 6.24. The inclusion ${}_{ELG}\overset{\varepsilon}{\Phi} \subseteq {}_{IE2GFA}\overset{\varepsilon}{\Phi}_{init\text{-}even}$ follows from Lemma 6.26. Hence, the theorem holds. □

6.4 Summary

Figure 6.7 summarizes the achieved results concerning the computational restrictions of input-erasing two-way finite automata studied in this chapter.

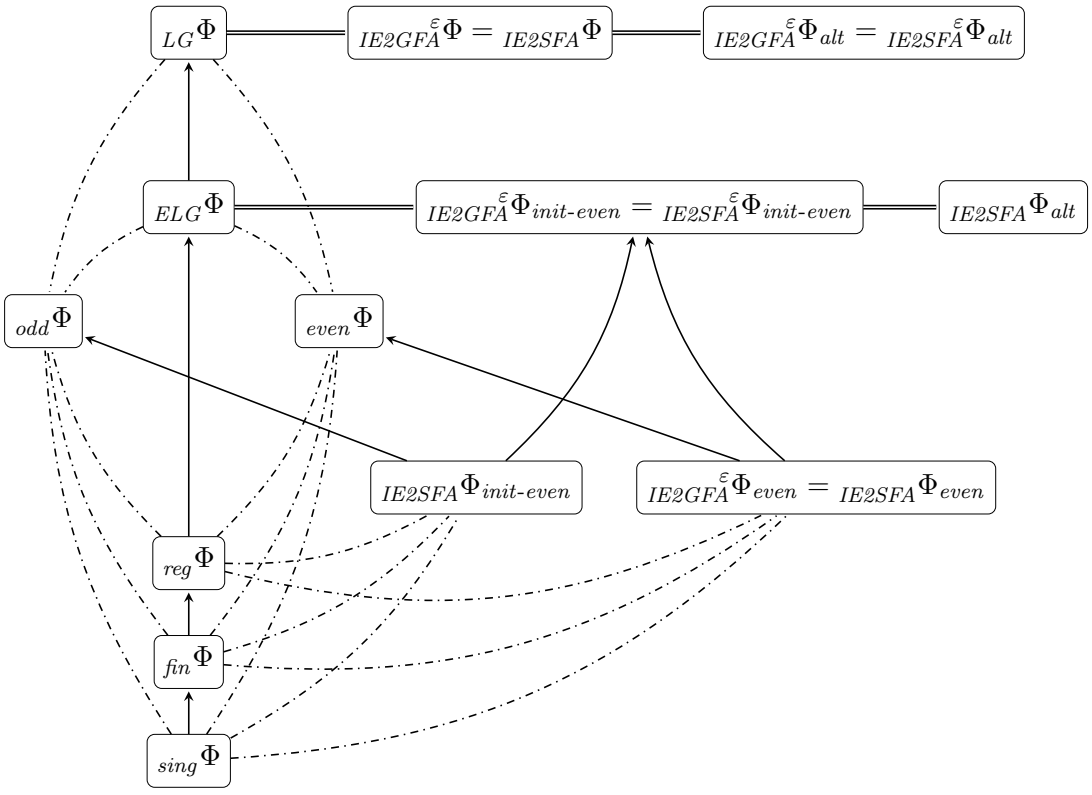

Figure 6.7: Relations between the language families of computationally restricted IE2GFAs and IE2SFAs and some other language families. A double line denotes equality, a solid arrow denotes proper inclusion, and a dash-dotted line denotes incomparability.

Chapter 7

Input-Related Restrictions

This chapter studies input-related restrictions of IE2GFAs. More specifically, it investigates the accepting power of these automata working under the assumption that their input strings or their parts belong to languages from some prescribed language families, such as the regular and linear language families. Theorems 7.1 and 7.2 show that regular-based input restrictions give rise to no increase in the power of IE2GFAs. Theorems 7.3 and 7.4 demonstrate that regular-based input restrictions can even lead to a decrease in the power of IE2GFAs to that of ordinary GFAs. These results are of some interest only when compared to the investigation of similar restrictions placed upon other rewriting systems, in which these restrictions give rise to a significant increase in their power. For instance, most selective grammars with regular-based selectors, which restrict the rewritten strings, are as strong as Turing machines (see Chapter 10 in [11] for a summary). In view of this increase in power in terms of other rewriting mechanisms, at a glance, we might hastily expect analogical results in terms of IE2GFAs, but the present chapter demonstrates that this is not the case. Finally, however, in Theorem 7.5, we show that linear-based input restrictions can increase the power of IE2GFAs.

Note that since ${}_{IE2GFA}^{\varepsilon}\Phi = {}_{IE2SFA}\Phi$, we can, without any loss of generality, work with ε-free IE2SFAs instead of IE2GFAs in Theorems 7.1 through 7.4.

Theorem 7.1. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $A, B \subseteq \Sigma^*$ be regular. Then, there exists an IE2SFA M' such that

$$L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\},$$

so $L(M')$ is linear.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $A, B \subseteq \Sigma^*$ be regular. Let $A = L(M_1)$ and $B = L(M_2)$, where $M_i = (Q_i, \Sigma, R_i, s_i, F_i)$ is an ε-free SFA for all $i \in \{1, 2\}$. From M, M_1, and M_2, we construct an IE2SFA $M' = (Q', \Sigma, R', s', F')$ such that $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$. Introduce a new symbol s'—the start state of M'. Set $\hat{Q} = \{\langle qq_1q_2\rangle \mid q \in Q, q_i \in Q_i, i \in \{1, 2\}\}$. Without any loss of generality, assume that $s' \notin \hat{Q}$. Set $Q' = \hat{Q} \cup \{s'\}$ and $F' = \{\langle fs_1f_2\rangle \mid f \in F, f_2 \in F_2\}$. Initially, set $R' = \varnothing$. Then, extend R' by performing steps (1) through (3), given next.

(1) For each $f_1 \in F_1$, add $s' \to \langle sf_1s_2\rangle$ to R'.

(2) For each $ap \to q \in R$ and $q_1a \to p_1 \in R_1$, where $p, q \in Q$, $p_1, q_1 \in Q_1$, and $a \in \Sigma$, add $a\langle pp_1q_2\rangle \to \langle qq_1q_2\rangle$ to R' for all $q_2 \in Q_2$.

(3) For each $pa \to q \in R$ and $p_2 a \to q_2 \in R_2$, where $p, q \in Q$, $p_2, q_2 \in Q_2$, and $a \in \Sigma$, add $\langle pq_1p_2\rangle a \to \langle qq_1q_2\rangle$ to R' for all $q_1 \in Q_1$.

Basic Idea. M', in effect, works in a two-directional way. To the right, it simulates a computation made by M and, simultaneously, a computation made by M_2 (see step (3)). To the left, it simulates a computation made by M and, simultaneously, a computation made by M_1 in reverse (see step (2)). Consider step (1) to see that M' accepts its input if and only if all the three automata—M, M_1, and M_2—accept their inputs as well, so $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$.

Let us now establish $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$ formally. We start by proving the following claim.

Claim 7.1.A. For all $u, v \in \Sigma^*$, $p, q \in Q$, $p_1, q_1 \in Q_1$, and $p_2, q_2 \in Q_2$,

$$u\langle pp_1p_2\rangle v \Rightarrow^* \langle qq_1q_2\rangle \text{ in } M' \text{ iff } upv \Rightarrow^* q \text{ in } M, q_1u \Rightarrow^* p_1 \text{ in } M_1, \text{and } p_2v \Rightarrow^* q_2 \text{ in } M_2.$$

Proof of Claim 7.1.A. First, we establish the *only if* part of this equivalence. By induction on the number of moves $i \geq 0$, we prove that $u\langle pp_1p_2\rangle v \Rightarrow^i \langle qq_1q_2\rangle$ in M' implies $upv \Rightarrow^* q$ in M, $q_1u \Rightarrow^* p_1$ in M_1, and $p_2v \Rightarrow^* q_2$ in M_2.

Basis. Let $i = 0$, so $u\langle pp_1p_2\rangle v \Rightarrow^0 \langle qq_1q_2\rangle$ in M'. Then, $p = q$, $p_1 = q_1$, $p_2 = q_2$, and $uv = \varepsilon$. Clearly, $p \Rightarrow^0 p$ in M, $p_1 \Rightarrow^0 p_1$ in M_1, and $p_2 \Rightarrow^0 p_2$ in M_2, so the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M', for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $u\langle pp_1p_2\rangle v \Rightarrow^{j+1} \langle qq_1q_2\rangle$ in M'. Let this computation start with the application of a rule of the form

$$a\langle pp_1p_2\rangle \to \langle oo_1p_2\rangle$$

from R', where $o \in Q$, $o_1 \in Q_1$, and $a \in \Sigma$. Thus, we can express $u\langle pp_1p_2\rangle v \Rightarrow^{j+1} \langle qq_1q_2\rangle$ as

$$u'a\langle pp_1p_2\rangle v \Rightarrow u'\langle oo_1p_2\rangle v \Rightarrow^j \langle qq_1q_2\rangle$$

in M', where $u'a = u$. By the induction hypothesis, $u'ov \Rightarrow^* q$ in M, $q_1u' \Rightarrow^* o_1$ in M_1, and $p_2v \Rightarrow^* q_2$ in M_2. Since step (2) constructs $a\langle pp_1p_2\rangle \to \langle oo_1p_2\rangle \in R'$ from $o_1a \to p_1 \in R_1$ and $ap \to o \in R$,

$$u'apv \Rightarrow u'ov \Rightarrow^* q$$

in M and

$$q_1u'a \Rightarrow^* o_1a \Rightarrow p_1$$

in M_1. Because $u'a = u$, $upv \Rightarrow^* q$ in M and $q_1u \Rightarrow^* p_1$ in M_1.

In the case that the computation $u\langle pp_1p_2\rangle v \Rightarrow^{j+1} \langle qq_1q_2\rangle$ in M' starts with the application of a rule of the form $\langle pp_1p_2\rangle a \to \langle op_1o_2\rangle$ from R', where $o \in Q$, $o_2 \in Q_2$, and $a \in \Sigma$, we can proceed analogously.

Thus, the induction step is completed.

Now, we establish the *if* part of the equivalence stated in Claim 7.1.A, so we show that $upv \Rightarrow^i q$ in M, $q_1u \Rightarrow^j p_1$ in M_1, and $p_2v \Rightarrow^k q_2$ in M_2, where $j + k = i$, implies $u\langle pp_1p_2\rangle v \Rightarrow^* \langle qq_1q_2\rangle$ in M' by induction on the number of moves $i \geq 0$.

Basis. Let $i = 0$, so $j = 0$, $k = 0$, $upv \Rightarrow^0 q$ in M, $q_1u \Rightarrow^0 p_1$ in M_1, and $p_2v \Rightarrow^0 q_2$ in M_2. Then, $p = q$, $p_1 = q_1$, $p_2 = q_2$, and $uv = \varepsilon$. Since $\langle pp_1p_2\rangle \Rightarrow^0 \langle pp_1p_2\rangle$ in M', the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than l moves in M, for some $l \in \mathbb{N}_0$.

Induction Step. Consider any $upv \Rightarrow^{l+1} q$ in M, $q_1u \Rightarrow^{m+1} p_1$ in M_1, and $p_2v \Rightarrow^{n} q_2$ in M_2, where $m+n=l$. Let $upv \Rightarrow^{l+1} q$ in M start with the application of a rule of the form

$$ap \to o$$

from R and $q_1u \Rightarrow^{m+1} p_1$ in M_1 end with the application of a rule of the form

$$o_1a \to p_1$$

from R_1, where $o \in Q$, $o_1 \in Q_1$, and $a \in \Sigma$. Express $upv \Rightarrow^{l+1} q$ as

$$u'apv \Rightarrow u'ov \Rightarrow^{l} q$$

in M and $q_1u \Rightarrow^{m+1} p_1$ as

$$q_1u'a \Rightarrow^{m} o_1a \Rightarrow p_1$$

in M_1, where $u'a = u$. By the induction hypothesis, we have $u'\langle oo_1p_2\rangle v \Rightarrow^* \langle qq_1q_2\rangle$ in M'. From $ap \to o \in R$ and $o_1a \to p_1 \in R_1$, step (2) constructs $a\langle pp_1p_2\rangle \to \langle oo_1p_2\rangle \in R'$. Thus, M' makes

$$u'a\langle pp_1p_2\rangle v \Rightarrow u'\langle oo_1p_2\rangle v \Rightarrow^* \langle qq_1q_2\rangle.$$

Since $u'a = u$, $u\langle pp_1p_2\rangle v \Rightarrow^* \langle qq_1q_2\rangle$ in M'.

Next, consider any $upv \Rightarrow^{l+1} q$ in M, $q_1u \Rightarrow^{m} p_1$ in M_1, and $p_2v \Rightarrow^{n+1} q_2$ in M_2, where $m+n=l$. Let $upv \Rightarrow^{l+1} q$ in M start with the application of a rule of the form $pa \to o$ from R and $p_2v \Rightarrow^{n+1} q_2$ in M_2 start with the application of a rule of the form $p_2a \to o_2$ from R_2, where $o \in Q$, $o_2 \in Q_2$, and $a \in \Sigma$. Then, proceed by analogy with the previous case.

Thus, the induction step is completed, and Claim 7.1.A holds.

Consider Claim 7.1.A for $p = s$, $q_1 = s_1$, and $p_2 = s_2$. At this point, for all $u, v \in \Sigma^*$, $q \in Q$, $p_1 \in Q_1$, and $q_2 \in Q_2$, $u\langle sp_1s_2\rangle v \Rightarrow^* \langle qs_1q_2\rangle$ in M' iff $usv \Rightarrow^* q$ in M, $s_1u \Rightarrow^* p_1$ in M_1, and $s_2v \Rightarrow^* q_2$ in M_2. As follows from the construction of R', M' starts every computation by applying a rule of the form $s' \to \langle sf_1s_2\rangle$ with $f_1 \in F_1$. Consequently, $us'v \Rightarrow u\langle sf_1s_2\rangle v \Rightarrow^* \langle qs_1q_2\rangle$ in M' iff $usv \Rightarrow^* q$ in M, $s_1u \Rightarrow^* f_1$ in M_1, and $s_2v \Rightarrow^* q_2$ in M_2. Considering this equivalence for $q = f$ and $q_2 = f_2$, where $f \in F$ and $f_2 \in F_2$, we obtain $us'v \Rightarrow u\langle sf_1s_2\rangle v \Rightarrow^* \langle fs_1f_2\rangle$ in M' iff $usv \Rightarrow^* f$ in M, $s_1u \Rightarrow^* f_1$ in M_1, and $s_2v \Rightarrow^* f_2$ in M_2. Recall that $F' = \{\langle fs_1f_2\rangle \mid f \in F, f_2 \in F_2\}$. Therefore, $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in L(M_1), v \in L(M_2)\}$. Since $L(M_1) = A$ and $L(M_2) = B$, $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$, so Theorem 7.1 holds. □

Theorem 7.2. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $A \subseteq \Sigma^*$ be regular. Then, there exists an IE2SFA M' satisfying

$$L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, uv \in A\},$$

so $L(M')$ is linear.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $A \subseteq \Sigma^*$ be regular. Let $A = L(\hat{M})$, where $\hat{M} = (\hat{Q}, \Sigma, \hat{R}, \hat{s}, \hat{F})$ is an ε-free SFA. From M and $\hat{M}$, we next construct an IE2SFA $M' = (Q', \Sigma, R', s', F')$ such that $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, uv \in A\}$. Introduce a new symbol s'—the start state of M'. Set $\bar{Q} = \{\langle q\hat{p}\hat{q}\rangle \mid q \in Q, \hat{p}, \hat{q} \in \hat{Q}\}$. Without any loss of generality, assume that $s' \notin \bar{Q}$. Set $Q' = \bar{Q} \cup \{s'\}$ and $F' = \{\langle f\hat{s}\hat{f}\rangle \mid f \in F, \hat{f} \in \hat{F}\}$. Initially, set $R' = \varnothing$. Then, extend R' by performing the following steps:

(1) for each $\hat{q} \in \hat{Q}$, add $s' \to \langle s\hat{q}\hat{q}\rangle$ to R';

(2) for each $ap \to q \in R$ and $\hat{q}a \to \hat{p} \in \hat{R}$, where $p, q \in Q$, $\hat{p}, \hat{q} \in \hat{Q}$, and $a \in \Sigma$, add $a\langle p\hat{p}\hat{o}\rangle \to \langle q\hat{q}\hat{o}\rangle$ to R' for all $\hat{o} \in \hat{Q}$;

(3) for each $pa \to q \in R$ and $\hat{p}a \to \hat{q} \in \hat{R}$, where $p, q \in Q$, $\hat{p}, \hat{q} \in \hat{Q}$, and $a \in \Sigma$, add $\langle p\hat{o}\hat{p}\rangle a \to \langle q\hat{o}\hat{q}\rangle$ to R' for all $\hat{o} \in \hat{Q}$.

Basic Idea. As can be seen, M' works in a two-directional way. To the right, it simulates a computation made by M and, simultaneously, a computation made by $\hat{M}$ (see step (3)). To the left, it simulates a computation made by M and, simultaneously, a computation made by $\hat{M}$ in reverse (see step (2)). Considering step (1), observe that M' accepts its input if and only if both M and $\hat{M}$ accept their inputs, too, so $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, uv \in A\}$.

Complete this proof by analogy with the proof of Theorem 7.1. □

Theorem 7.3. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, $A \subseteq \Sigma^*$ be finite, and $B \subseteq \Sigma^*$ be regular. Then, there exists an SFA M' such that

$$L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\},$$

so $L(M')$ is regular.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, $A \subseteq \Sigma^*$ be finite, and $B \subseteq \Sigma^*$ be regular. Next, we construct an SFA $M' = (Q', \Sigma, R', s', F')$ such that $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$. Let $n = \max\{|x| \mid x \in A\}$. Let $\hat{M} = (\hat{Q}, \Sigma, \hat{R}, \hat{s}, \hat{F})$ be an ε-free SFA such that $L(\hat{M}) = B$. Set $Q' = \{\langle x\rangle, \langle xq\hat{q}\rangle \mid x \in \Sigma^*, 0 \leq |x| \leq n, q \in Q, \hat{q} \in \hat{Q}\}$, $s' = \langle\varepsilon\rangle$, and $F' = \{\langle f\hat{f}\rangle \mid f \in F, \hat{f} \in \hat{F}\}$. R' is constructed in the following way:

(1) for each $\langle x\rangle, \langle xa\rangle \in Q'$, where $x \in \Sigma^*$ and $a \in \Sigma$, add $\langle x\rangle a \to \langle xa\rangle$ to R';

(2) for each $x \in A$, add $\langle x\rangle \to \langle xs\hat{s}\rangle$ to R';

(3) for each $ap \to q \in R$ and $\langle xap\hat{q}\rangle, \langle xq\hat{q}\rangle \in Q'$, where $p, q \in Q$, $\hat{q} \in \hat{Q}$, $a \in \Sigma$ and $x \in \Sigma^*$, add $\langle xap\hat{q}\rangle \to \langle xq\hat{q}\rangle$ to R';

(4) for each $pa \to q \in R$, $\hat{p}a \to \hat{q} \in \hat{R}$, and $\langle xp\hat{p}\rangle, \langle xq\hat{q}\rangle \in Q'$, where $p, q \in Q$, $\hat{p}, \hat{q} \in \hat{Q}$, and $x \in \Sigma^*$, add $\langle xp\hat{p}\rangle a \to \langle xq\hat{q}\rangle$ to R'.

Basic Idea. M' starts every computation by reading a string from the language A and recording it into its current state (see step (1)). After this initial phase, M' begins to simulate computations made by M and $\hat{M}$ (see step (2)). All left moves made by M are simulated by M' exclusively within its states by successively erasing symbols from the recorded string (see step (3)). All right moves made by M and, simultaneously, a computation made by $\hat{M}$ are simulated by M' simply by processing the remaining part of the input tape (see step (4)).

Next, we demonstrate $L(M') = \{uv \mid usv \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B\}$ rigorously. We start by proving the following claim.

Claim 7.3.A. For all $u, v \in \Sigma^*$, $p, q \in Q$, and $\hat{p}, \hat{q} \in \hat{Q}$ such that $0 \leq |u| \leq n$,

$$\langle up\hat{p}\rangle v \Rightarrow^* \langle q\hat{q}\rangle \text{ in } M' \text{ iff } upv \Rightarrow^* q \text{ in } M \text{ and } \hat{p}v \Rightarrow^* \hat{q} \text{ in } \hat{M}.$$

Proof of Claim 7.3.A. First, we establish the *only if* part of this equivalence. By induction on the number of moves $i \geq 0$, we show that $\langle upp\hat{p}\rangle v \Rightarrow^i \langle q\hat{q}\rangle$ in M' implies $upv \Rightarrow^* q$ in M and $\hat{p}v \Rightarrow^* \hat{q}$ in $\hat{M}$.

Basis. Let $i = 0$, so $\langle upp\hat{p}\rangle v \Rightarrow^0 \langle q\hat{q}\rangle$ in M'. Then, $p = q$, $\hat{p} = \hat{q}$, and $uv = \varepsilon$. Clearly, $p \Rightarrow^0 p$ in M and $\hat{p} \Rightarrow^0 \hat{p}$ in $\hat{M}$, so the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M', for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $\langle upp\hat{p}\rangle v \Rightarrow^{j+1} \langle q\hat{q}\rangle$ in M'. Let this computation start with the application of a rule of the form

$$\langle u'app\hat{p}\rangle \rightarrow \langle u'op\hat{p}\rangle$$

from R', where $o \in Q$, $u'a = u$, and $a \in \Sigma$. Thus, we can express $\langle upp\hat{p}\rangle v \Rightarrow^{j+1} \langle q\hat{q}\rangle$ as

$$\langle u'app\hat{p}\rangle v \Rightarrow \langle u'op\hat{p}\rangle v \Rightarrow^j \langle q\hat{q}\rangle$$

in M'. Since $\langle u'op\hat{p}\rangle v \Rightarrow^j \langle q\hat{q}\rangle$ in M', by the induction hypothesis, $u'ov \Rightarrow^* q$ in M and $\hat{p}v \Rightarrow^* \hat{q}$ in $\hat{M}$. Step (3) constructs $\langle u'app\hat{p}\rangle \rightarrow \langle u'op\hat{p}\rangle \in R'$ from $ap \rightarrow o \in R$, so

$$u'apv \Rightarrow u'ov \Rightarrow^* q$$

in M. Since $u'a = u$, we have $upv \Rightarrow^* q$ in M.

Next, suppose that the computation $\langle upp\hat{p}\rangle v \Rightarrow^{j+1} \langle q\hat{q}\rangle$ in M' starts with the application of a rule of the form

$$\langle upp\hat{p}\rangle a \rightarrow \langle uo\hat{o}\rangle$$

from R', where $o \in Q$, $\hat{o} \in \hat{Q}$, and $a \in \Sigma$. Express $\langle upp\hat{p}\rangle v \Rightarrow^{j+1} \langle q\hat{q}\rangle$ as

$$\langle upp\hat{p}\rangle av' \Rightarrow \langle uo\hat{o}\rangle v' \Rightarrow^j \langle q\hat{q}\rangle$$

in M', where $v'a = v$. By the induction hypothesis, $uov' \Rightarrow^* q$ in M and $\hat{o}v' \Rightarrow^* \hat{q}$ in $\hat{M}$. Since step (4) constructs $\langle upp\hat{p}\rangle a \rightarrow \langle uo\hat{o}\rangle \in R'$ from $pa \rightarrow o \in R$ and $\hat{p}a \rightarrow \hat{o} \in \hat{R}$, it follows that

$$upav' \Rightarrow uov' \Rightarrow^* q$$

in M and

$$\hat{p}av' \Rightarrow \hat{o}v' \Rightarrow^* \hat{q}$$

in $\hat{M}$. Because $av' = v$, $upv \Rightarrow^* q$ in M and $\hat{p}v \Rightarrow^* \hat{q}$ in $\hat{M}$.

Thus, the induction step is completed.

Now, we establish the *if* part of the equivalence stated in Claim 7.3.A, so we show that $upv \Rightarrow^i q$ in M and $\hat{p}v \Rightarrow^j \hat{q}$ in $\hat{M}$, where $j \leq i$, implies $\langle upp\hat{p}\rangle v \Rightarrow^* \langle q\hat{q}\rangle$ in M' by induction on the number of moves $i \geq 0$.

Basis. Let $i = 0$, so $j = 0$, $upv \Rightarrow^0 q$ in M, and $\hat{p}v \Rightarrow^0 \hat{q}$ in $\hat{M}$. Then, $p = q$, $\hat{p} = \hat{q}$, and $uv = \varepsilon$. Since $\langle p\hat{p}\rangle \Rightarrow^0 \langle p\hat{p}\rangle$ in M', the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than k moves in M, for some $k \in \mathbb{N}_0$.

Induction Step. Consider any $upv \Rightarrow^{k+1} q$ in M and $\hat{p}v \Rightarrow^{l} \hat{q}$ in $\hat{M}$, where $l \leq k$. Let $upv \Rightarrow^{k+1} q$ in M start with the application of a rule of the form

$$ap \to o$$

from R, where $o \in Q$ and $a \in \Sigma$. Then, express $upv \Rightarrow^{k+1} q$ as

$$u'apv \Rightarrow u'ov \Rightarrow^{k} q$$

in M, where $u = u'a$. Since $u'ov \Rightarrow^{k} q$ in M and $\hat{p}v \Rightarrow^{l} \hat{q}$ in $\hat{M}$, by the induction hypothesis, $\langle u'o\hat{p}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$ in M'. From $ap \to o \in R$, step (3) constructs $\langle u'ap\hat{p}\rangle \to \langle u'o\hat{p}\rangle \in R'$, so

$$\langle u'ap\hat{p}\rangle v \Rightarrow \langle u'o\hat{p}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$$

in M'. Because $u'a = u$, $\langle up\hat{p}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$ in M'.

Next, consider any $upv \Rightarrow^{k+1} q$ in M and $\hat{p}v \Rightarrow^{l+1} \hat{q}$ in $\hat{M}$ with $l \leq k$. Let $upv \Rightarrow^{k+1} q$ in M start with the application of a rule of the form

$$pa \to o$$

from R and $\hat{p}v \Rightarrow^{l+1} \hat{q}$ in $\hat{M}$ start with the application of a rule of the form

$$\hat{p}a \to \hat{o}$$

from $\hat{R}$, where $o \in Q$, $\hat{o} \in \hat{Q}$, and $a \in \Sigma$. Express $upv \Rightarrow^{k+1} q$ as

$$upav' \Rightarrow uov' \Rightarrow^{k} q$$

in M and $\hat{p}v \Rightarrow^{l+1} \hat{q}$ as

$$\hat{p}av' \Rightarrow \hat{o}v' \Rightarrow^{l} \hat{q}$$

in $\hat{M}$, where $av' = v$. By the induction hypothesis, $\langle uo\hat{o}\rangle v' \Rightarrow^{*} \langle q\hat{q}\rangle$ in M'. From $pa \to o \in R$ and $\hat{p}a \to \hat{o} \in \hat{R}$, step (4) constructs $\langle up\hat{p}\rangle a \to \langle uo\hat{o}\rangle \in R'$, so

$$\langle up\hat{p}\rangle av' \Rightarrow \langle uo\hat{o}\rangle v' \Rightarrow^{*} \langle q\hat{q}\rangle$$

in M'. Since $av' = v$, $\langle up\hat{p}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$ in M'.

Thus, the induction step is completed, and Claim 7.3.A holds.

Considering Claim 7.3.A for $p = s$ and $p' = s'$, we see that for all $u, v \in \Sigma^{*}$, $q \in Q$, and $\hat{q} \in \hat{Q}$ such that $0 \leq |u| \leq n$, $\langle us\hat{s}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$ in M' iff $usv \Rightarrow^{*} q$ in M and $\hat{s}v \Rightarrow^{*} \hat{q}$ in $\hat{M}$. As follows from the construction of R', M' starts every accepting computation by a sequence of moves of the form $\langle\varepsilon\rangle \Rightarrow^{*} \langle x\rangle \Rightarrow \langle xs\hat{s}\rangle$, where $x \in A$. Consequently, $\langle\varepsilon\rangle v \Rightarrow^{*} \langle u\rangle v \Rightarrow \langle us\hat{s}\rangle v \Rightarrow^{*} \langle q\hat{q}\rangle$ in M' iff $usv \Rightarrow^{*} q$ in M, $\hat{s}v \Rightarrow^{*} \hat{q}$ in $\hat{M}$, and $u \in A$. Now, consider this equivalence for $q = f$ and $\hat{q} = \hat{f}$, where $f \in F$ and $\hat{f} \in \hat{F}$. That is, $\langle\varepsilon\rangle v \Rightarrow^{*} \langle u\rangle v \Rightarrow \langle us\hat{s}\rangle v \Rightarrow^{*} \langle f\hat{f}\rangle$ in M' iff $usv \Rightarrow^{*} f$ in M, $\hat{s}v \Rightarrow^{*} \hat{f}$ in $\hat{M}$, and $u \in A$. Since $F' = \{\langle f\hat{f}\rangle \mid f \in F, \hat{f} \in \hat{F}\}$, it follows that $L(M') = \{uv \mid usv \Rightarrow^{*} f \text{ in } M, f \in F, u \in A, v \in L(\hat{M})\}$. Thus, given that $L(\hat{M}) = B$, we have $L(M') = \{uv \mid usv \Rightarrow^{*} f \text{ in } M, f \in F, u \in A, v \in B\}$. Therefore, Theorem 7.3 holds. □

Theorem 7.4. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA, and let $A, B, C \subseteq \Sigma^{*}$ be regular. Then, there exists an SFA M' such that

$$L(M') = \{v \mid usvw \Rightarrow^{*} f \text{ in } M, f \in F, u \in A, v \in B, w \in C\},$$

so $L(M')$ is regular.

Proof. Let $M = (Q, \Sigma, R, s, F)$ be an ε-free IE2SFA and $A, B, C \subseteq \Sigma^*$ be regular. Let $A = L(M_1)$, $B = L(M_2)$, and $C = L(M_3)$, where $M_i = (Q_i, \Sigma_i, R_i, s_i, F_i)$ is an ε-free SFA for all $i \in \{1, 2, 3\}$. From M, M_1, M_2, and M_3, we construct an SFA $M' = (Q', \Sigma, R', s', F')$ satisfying $L(M') = \{v \mid usvw \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B, w \in C\}$. Introduce a new symbol s'—the start state of M'. Set $\hat{Q} = \{\langle qq_1q_2s_31\rangle, \langle qq_1f_2q_32\rangle \mid q \in Q, q_i \in Q_i, i \in \{1, 2, 3\}, f_2 \in F_2\}$. Without any loss of generality, assume that $s' \notin \hat{Q}$. Set $Q' = \hat{Q} \cup \{s'\}$ and $F' = \{\langle fs_1f_2f_32\rangle \mid f \in F, f_i \in F_i, i \in \{2, 3\}\}$. R' is constructed by performing steps (1) through (5), given next.

(1) For each $f_1 \in F_1$, add $s' \to \langle sf_1s_2s_31\rangle$ to R'.

(2) For each $q \in Q$, $q_1 \in Q_1$, and $f_2 \in F_2$, add $\langle qq_1f_2s_31\rangle \to \langle qq_1f_2s_32\rangle$ to R'.

(3) For each $ap \to q \in R$ and $q_1a \to p_1 \in R_1$, where $p, q \in Q$, $p_1, q_1 \in Q_1$, and $a \in \Sigma$, add $\langle pp_1q_2s_31\rangle \to \langle qq_1q_2s_31\rangle$ and $\langle pp_1f_2q_32\rangle \to \langle qq_1f_2q_32\rangle$ to R' for all $q_2 \in Q_2$, $q_3 \in Q_3$, and $f_2 \in F_2$.

(4) For each $pa \to q \in R$ and $p_2a \to q_2 \in R_2$, where $p, q \in Q$, $p_2, q_2 \in Q_2$, and $a \in \Sigma$, add $\langle pq_1p_2s_31\rangle a \to \langle qq_1q_2s_31\rangle$ to R' for all $q_1 \in Q_1$.

(5) For each $pa \to q \in R$ and $p_3a \to q_3 \in R_3$, where $p, q \in Q$, $p_3, q_3 \in Q_3$, and $a \in \Sigma$, add $\langle pq_1f_2p_32\rangle \to \langle qq_1f_2q_32\rangle$ to R' for all $q_1 \in Q_1$ and $f_2 \in F_2$.

Basic Idea. M' works in two phases. During the first phase, it simulates all right moves made by M and, simultaneously, a computation made by M_2 by processing the input tape (see step (4)). During the second phase, it simulates all right moves made by M and, simultaneously, a computation made by M_3, entirely without reading any input symbols (see step (5)). In addition, during both of these phases, M' simulates all left moves made by M and, simultaneously, a computation made by M_1 in reverse, again, using only ε-rules (see step (3)). Finally, consider steps (1) and (2) to see that M' accepts its input if and only if M, M_1, M_2, and M_3 accept their inputs as well, so $L(M') = \{v \mid usvw \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B, w \in C\}$.

To establish $L(M') = \{v \mid usvw \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B, w \in C\}$ formally, we first prove the following two claims.

Claim 7.4.A. For all $v \in \Sigma^*$, $p, q \in Q$, $p_1, q_1 \in Q_1$, and $p_2, q_2 \in Q_2$,

$$\langle pp_1p_2s_31\rangle v \Rightarrow^* \langle qq_1q_2s_31\rangle \text{ in } M' \text{ iff there is } x \in \Sigma^* \text{ such that } \begin{cases} xpv \Rightarrow^* q \text{ in } M, \\ q_1x \Rightarrow^* p_1 \text{ in } M_1, \text{ and} \\ p_2v \Rightarrow^* q_2 \text{ in } M_2. \end{cases}$$

Proof of Claim 7.4.A. First, we establish the *only if* part of this equivalence. By induction on the number of moves $i \geq 0$, we show that $\langle pp_1p_2s_31\rangle v \Rightarrow^i \langle qq_1q_2s_31\rangle$ in M' implies that there is $x \in \Sigma^*$ such that $xpv \Rightarrow^* q$ in M, $q_1x \Rightarrow^* p_1$ in M_1, and $p_2v \Rightarrow^* q_2$ in M_2.

Basis. Let $i = 0$, so $\langle pp_1p_2s_31\rangle v \Rightarrow^0 \langle qq_1q_2s_31\rangle$ in M'. Then, $p = q$, $p_1 = q_1$, $p_2 = q_2$, and $v = \varepsilon$. Clearly, $p \Rightarrow^0 p$ in M, $p_1 \Rightarrow^0 p_1$ in M_1, and $p_2 \Rightarrow^0 p_2$ in M_2, so the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than j moves in M', for some $j \in \mathbb{N}_0$.

Induction Step. Consider any computation of the form $\langle pp_1p_2s_31\rangle v \Rightarrow^{j+1} \langle qq_1q_2s_31\rangle$ in M'. Let this computation start with the application of a rule of the form

$$\langle pp_1p_2s_31\rangle \rightarrow \langle oo_1p_2s_31\rangle$$

from R', where $o \in Q$ and $o_1 \in Q_1$. Now, express $\langle pp_1p_2s_31\rangle v \Rightarrow^{j+1} \langle qq_1q_2s_31\rangle$ as

$$\langle pp_1p_2s_31\rangle v \Rightarrow \langle oo_1p_2s_31\rangle v \Rightarrow^{j} \langle qq_1q_2s_31\rangle$$

in M'. Since $\langle oo_1p_2s_31\rangle v \Rightarrow^{j} \langle qq_1q_2s_31\rangle$ in M', by the induction hypothesis, $x'ov \Rightarrow^* q$ in M, $q_1x' \Rightarrow^* o_1$ in M_1, and $p_2v \Rightarrow^* q_2$ in M_2, for some $x' \in \Sigma^*$. Step (3) constructs $\langle pp_1p_2s_31\rangle \rightarrow \langle oo_1p_2s_31\rangle \in R'$ from $ap \rightarrow o \in R$ and $o_1a \rightarrow p_1 \in R_1$, for some $a \in \Sigma$, so

$$x'apv \Rightarrow x'ov \Rightarrow^* q$$

in M and

$$q_1x'a \Rightarrow^* o_1a \Rightarrow p_1$$

in M_1. Hence, assuming that $x = x'a$, we have $xpv \Rightarrow^* q$ in M and $q_1x \Rightarrow^* p_1$ in M_1.

If the computation $\langle pp_1p_2s_31\rangle v \Rightarrow^{j+1} \langle qq_1q_2s_31\rangle$ in M' starts with the application of a rule of the form $\langle pp_1p_2s_31\rangle a \rightarrow \langle op_1o_2s_31\rangle$ from R', where $o \in Q$, $o_2 \in Q_2$, and $a \in \Sigma$, proceed analogously.

Thus, the induction step is completed.

Next, we establish the *if* part of the equivalence stated in Claim 7.4.A. By induction on the number of moves $i \geq 0$, we prove that $xpv \Rightarrow^i q$ in M, $q_1x \Rightarrow^j p_1$ in M_1, and $p_2v \Rightarrow^k q_2$ in M_2, where $j + k = i$, implies $\langle pp_1p_2s_31\rangle v \Rightarrow^* \langle qq_1q_2s_31\rangle$ in M'.

Basis. Let $i = 0$, so $j = 0$, $k = 0$, $xpv \Rightarrow^0 q$ in M, $q_1x \Rightarrow^0 p_1$ in M_1, and $p_2v \Rightarrow^0 q_2$ in M_2. Then, $p = q$, $p_1 = q_1$, $p_2 = q_2$, and $xv = \varepsilon$. Since $\langle pp_1p_2s_31\rangle v \Rightarrow^0 \langle pp_1p_2s_31\rangle$ in M', the basis holds true.

Induction Hypothesis. Assume that the implication holds for all computations consisting of no more than l moves in M, for some $l \in \mathbb{N}_0$.

Induction Step. Consider any $xpv \Rightarrow^{l+1} q$ in M, $q_1x \Rightarrow^{m+1} p_1$ in M_1, and $p_2v \Rightarrow^n q_2$ in M_2, where $m + n = l$. Let $xpv \Rightarrow^{l+1} q$ in M start with the application of a rule the form

$$ap \rightarrow o$$

from R and $q_1x \Rightarrow^{m+1} p_1$ in M_1 end with the application of a rule of the form

$$o_1a \rightarrow p_1$$

from R_1, where $o \in Q$, $o_1 \in Q_1$, and $a \in \Sigma$. Express $xpv \Rightarrow^{l+1} q$ as

$$x'apv \Rightarrow x'ov \Rightarrow^{l} q$$

in M and $q_1x \Rightarrow^{m+1} p_1$ as

$$q_1x'a \Rightarrow^{m} o_1a \Rightarrow p_1$$

in M_1, where $x'a = x$. By the induction hypothesis, $\langle oo_1p_2s_31\rangle v \Rightarrow^* \langle qq_1q_2s_31\rangle$ in M'. From $ap \rightarrow o \in R$ and $o_1a \rightarrow p_1 \in R_1$, step (3) constructs $\langle pp_1p_2s_31\rangle \rightarrow \langle oo_1p_2s_31\rangle \in R'$, so

$$\langle pp_1p_2s_31\rangle v \Rightarrow \langle oo_1p_2s_31\rangle v \Rightarrow^* \langle qq_1q_2s_31\rangle$$

in M'. Therefore, $\langle pp_1p_2s_31\rangle v \Rightarrow^* \langle qq_1q_2s_31\rangle$ in M'.

Next, consider any $xpv \Rightarrow^{l+1} q$ in M, $q_1x \Rightarrow^m p_1$ in M_1, and $p_2v \Rightarrow^{n+1} q_2$ in M_2, where $m+n=l$. Let $xpv \Rightarrow^{l+1} q$ in M start with the application of a rule of the form $pa \to o$ from R and $p_2v \Rightarrow^{n+1} q_2$ in M_2 start with the application of a rule of the form $p_2a \to o_2$ from R_2, where $o \in Q$, $o_2 \in Q_2$, and $a \in \Sigma$. Then, proceed by analogy with the previous case.

Thus, the induction step is completed, and Claim 7.4.A holds.

Claim 7.4.B. For all $q, o \in Q$, $q_1, o_1 \in Q_1$, $f_2 \in F_2$, and $q_3, o_3 \in Q_3$,

$$\langle qq_1f_2q_32\rangle \Rightarrow^* \langle oo_1f_2o_32\rangle \text{ in } M' \text{ iff there are } y, w \in \Sigma^* \text{ such that } \begin{cases} yqw \Rightarrow^* o \text{ in } M, \\ o_1y \Rightarrow^* q_1 \text{ in } M_1, \text{ and} \\ q_3w \Rightarrow^* o_3 \text{ in } M_3. \end{cases}$$

Proof of Claim 7.4.B. Prove this claim by analogy with the proof of Claim 7.4.A.

Observe that M' starts every accepting computation by applying a rule of the form $s' \to \langle sf_1s_2s_31\rangle$, where $f_1 \in F_1$, and also uses a rule of the form $\langle qq_1f_2s_31\rangle \to \langle qq_1f_2s_32\rangle$, where $q \in Q$, $q_1 \in Q_1$, and $f_2 \in F_2$, at some point during each such computation (see steps (1) and (2)). From these observations together with Claims 7.4.A and 7.4.B, it follows that for all $q, o \in Q$, $q_1, o_1 \in Q_1$, $o_3 \in Q_3$, $f_1 \in F_1$, $f_2 \in F_2$, and $v \in \Sigma^*$, $s'v \Rightarrow \langle sf_1s_2s_31\rangle v \Rightarrow^* \langle qq_1f_2s_31\rangle \Rightarrow \langle qq_1f_2s_32\rangle \Rightarrow^* \langle os_1f_2o_32\rangle$ in M' iff there are $u, w \in \Sigma^*$ such that $usvw \Rightarrow^* o$ in M, $s_1u \Rightarrow^* f_1$ in M_1, $s_2v \Rightarrow^* f_2$ in M_2, and $s_3w \Rightarrow^* o_3$ in M_3. Considering this equivalence for $o = f$ and $o_3 = f_3$, where $f \in F$ and $f_3 \in F_3$, we can see that $s'v \Rightarrow \langle sf_1s_2s_31\rangle v \Rightarrow^* \langle qq_1f_2s_31\rangle \Rightarrow \langle qq_1f_2s_32\rangle \Rightarrow^* \langle fs_1f_2f_32\rangle$ in M' iff there are $u, w \in \Sigma^*$ such that $usvw \Rightarrow^* f$ in M, $s_1u \Rightarrow^* f_1$ in M_1, $s_2v \Rightarrow^* f_2$ in M_2, and $s_3w \Rightarrow^* f_3$ in M_3. Recall that $F' = \{\langle fs_1f_2f_32\rangle \mid f \in F, f_i \in F_i, i \in \{2,3\}\}$. Hence, $L(M') = \{v \mid usvw \Rightarrow^* f \text{ in } M, f \in F, u \in L(M_1), v \in L(M_2), w \in L(M_3)\}$. As $L(M_1) = A$, $L(M_2) = B$, and $L(M_3) = C$, we have $L(M') = \{v \mid usvw \Rightarrow^* f \text{ in } M, f \in F, u \in A, v \in B, w \in C\}$. Therefore, Theorem 7.4 holds. □

Theorem 7.5. There exist an IE2GFA $M = (Q, \Sigma, R, s, F)$, $A \in {}_{reg}\Phi$, and $B \in {}_{LG}\Phi$ such that

$$\{uv \mid usv \Rightarrow^* f, f \in F, u \in A, v \in B\}$$

is not linear.

Proof. Consider the ε-free IE2SFA

$$M = (\{s, q, f\}, \{a, b, c\}, R, s, \{f\})$$

with $R = \{sb \to q, aq \to s, sc \to f, fc \to f\}$ (see Figure 7.1), $A = \{a\}^*$, and $B = \{b^nc^n \mid n \geq 0\}$. Clearly, $L(M) = \{a^nb^nc^m \mid m, n \geq 0\}$, $A \in {}_{reg}\Phi$, and $B \in {}_{LG}\Phi$. However, observe that the language $K = \{uv \mid usv \Rightarrow^* f, f \in F, u \in A, v \in B\} = \{a^nb^nc^n \mid n \geq 0\}$ is not linear. In fact, $K \in {}_{CSG}\Phi \setminus {}_{CFG}\Phi$ (see Example 2.21). Thus, the theorem holds. □

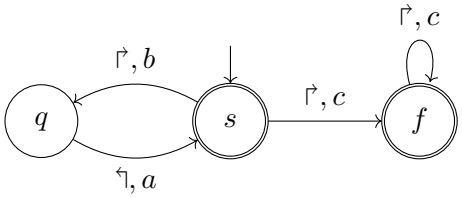

Figure 7.1: State diagram of the ε-free IE2SFA M from the proof of Theorem 7.5.

Chapter 8

Conclusion

The present thesis has proposed and studied new versions of two-way finite automata referred to as input-erasing two-way finite automata. In essence, they perform their computation just like the classical versions of these automata, except that (1) they erase the input symbols just like one-way finite automata do, and (2) they start their computation at any position on the input tape.

First, we recalled all the terminology necessary for this thesis, ranging from basic notions such as strings and languages to formal grammars and one-way finite automata. After this, we described the classical concept of two-way finite automata. We explained how they work and illustrated their features.

The key part of this thesis then introduced input-erasing two-way finite automata and investigated their properties. We formally defined these new automata, highlighted their key differences from the classical model, and demonstrated how they work. The main result is that these automata define the same language family as linear grammars. We proved this by showing that any such automaton can be transformed into an equivalent linear grammar and vice versa. We also demonstrated that general and simple variants of these automata, including those without ε-rules, are all equally powerful.

Next, we investigated three restrictions imposed on the way the proposed automata work—namely, alternating, even, and initialized even computations. We established relations between the language families resulting from these restrictions and demonstrated that under initialized even computation, these automata are as strong as even linear grammars.

Lastly, we discussed several restrictions placed upon the input of the proposed automata. We showed that those based on regular languages do not lead to any increase in their computational power. Some even reduce it to the regular language family. In contrast, however, linear-based input restrictions can extend the accepting capabilities of these automata even to some non-context-free languages.

As part of this thesis, we also implemented a program that simulates the newly introduced input-erasing two-way finite automata. It is designed as a console application and can simulate both their restricted (alternating, even, and initialized even) and unrestricted computations. The primary purpose of this program is to demonstrate how the proposed automata work and how they can behave in practice. The implementation details of this program and its manual can be found in Appendix A.

Since the proposed automata possess the same power as linear grammars and are thus stronger than classical one-way and two-way finite automata, they can be used for more complex analyses, especially for recognizing linear or palindromic patterns. For example, in bioinformatics, they could be easily used to identify complementary strands of DNA

molecules (see [37]). Further investigation of their possible applications in biological sequence analysis (see [16]) or in other fields of study could be an interesting subject for future research.

Although this thesis has established several fundamental results concerning input-erasing two-way finite automata and their restricted versions, there still remain several open problem areas to study. These include

(i) an investigation of more classical topics of automata theory, such as determinism and minimization;

(ii) a further investigation of input-related restrictions, for instance, in terms of subregular language families;

(iii) a conceptualization and investigation of input-erasing two-way finite automata in an alternative way by analogy with other modern concepts of automata, such as regulated and jumping versions (see [31, 32]); and

(iv) an introduction and investigation of other types of automata, such as pushdown automata (see [17]), conceptualized by analogy with the input-erasing two-way finite automata given in the present thesis.

Bibliography

[1] Amar, V. and Putzolu, G. On a family of linear grammars. *Information and Control*, 1964, vol. 7, no. 3, p. 283–291. ISSN 0019-9958.

[2] Ambainis, A. and Watrous, J. Two-way finite automata with quantum and classical states. *Theoretical Computer Science*, 2002, vol. 287, no. 1, p. 299–311. ISSN 0304-3975.

[3] Balcerzak, M. and Niwiński, D. Two-way deterministic automata with two reversals are exponentially more succinct than with one reversal. *Information Processing Letters*, 2010, vol. 110, no. 10, p. 396–398. ISSN 0020-0190.

[4] Birget, J.-C. Basic techniques for two-way finite automata. In: *Proceedings of the LITP Spring School on Theoretical Computer Science: Formal Properties of Finite Automata and Applications*. Berlin, Heidelberg: Springer-Verlag, 1987, p. 56–64. ISBN 978-3-540-51631-6.

[5] Birget, J.-C. Positional simulation of two-way automata: Proof of a conjecture of R. Kannan and generalizations. *Journal of Computer and System Sciences*, 1992, vol. 45, no. 2, p. 154–179. ISSN 0022-0000.

[6] Birget, J.-C. Two-way automata and length-preserving homomorphisms. *Mathematical systems theory*, june 1996, vol. 29, no. 3, p. 191–226. ISSN 1433-0490.

[7] Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 1986, C-35, no. 8, p. 677–691. ISSN 0018-9340.

[8] Chomsky, N. Three models for the description of language. *I.R.E. transactions on information theory*. The Institute of Radio Engineers, Inc, 1956, vol. 2, no. 3, p. 113–124. ISSN 0096-1000.

[9] Chomsky, N. On certain formal properties of grammars. *Information and control*. Elsevier B.V, 1959, vol. 2, no. 2, p. 137–167. ISSN 0019-9958.

[10] Chrobak, M. Finite automata and unary languages. *Theoretical Computer Science*, 1986, vol. 47, p. 149–158. ISSN 0304-3975.

[11] Dassow, J. and Paun, G. *Regulated Rewriting in Formal Language Theory*. Springer Berlin Heidelberg, 2011. Monographs in Theoretical Computer Science. An EATCS Series. ISBN 978-3-642-74934-6.

[12] Fazekas, S. Z.; Hoshi, K. and Yamamura, A. Two-way deterministic automata with jumping mode. *Theoretical Computer Science*, 2021, vol. 864, p. 92–102. ISSN 0304-3975.

[13] Frei, F.; Hromkovič, J.; Královič, R. and Královič, R. Two-Way Non-uniform Finite Automata. In: Moreira, N. and Reis, R., ed. *Developments in Language Theory*. Cham: Springer International Publishing, 2021, p. 155–166. ISBN 978-3-030-81508-0.

[14] Freivalds, R. Probabilistic two-way machines. In: Gruska, J. and Chytil, M., ed. *Mathematical Foundations of Computer Science 1981*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, p. 33–45. ISBN 978-3-540-38769-5.

[15] Grzegorz, R. and Salomaa, A. *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*. Springer Berlin, Heidelberg, 1997. ISBN 978-3-540-60420-4.

[16] Gusfield, D. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997. ISBN 978-0-521-58519-4.

[17] Hopcroft, J. E. and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979. Addison-Wesley Series in Computer Science and Information Processing. ISBN 978-0-201-02988-8.

[18] Hromkovič, J. and Schnitger, G. Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation. In: Baeten, J. C. M.; Lenstra, J. K.; Parrow, J. and Woeginger, G. J., ed. *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, p. 439–451. ISBN 978-3-540-45061-0.

[19] Jirásková, G. and Klíma, O. On linear languages recognized by deterministic biautomata. *Information and Computation*, july 2022, vol. 286, p. 1–22. ISSN 0890-5401. Article 104778.

[20] Kapoutsis, C. Removing Bidirectionality from Nondeterministic Finite Automata. In: Jędrzejowicz, J. and Szepietowski, A., ed. *Mathematical Foundations of Computer Science 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, p. 544–555. ISBN 978-3-540-31867-5.

[21] Kapoutsis, C. and Zakzok, M. Alternation in two-way finite automata. *Theoretical Computer Science*, 2021, vol. 870, p. 75–102. ISSN 0304-3975.

[22] Kapoutsis, C. A. Two-Way Automata Versus Logarithmic Space. *Theory of Computing Systems*, august 2014, vol. 55, no. 2, p. 421–447. ISSN 1433-0490.

[23] Kožár, T. and Meduna, A. *Automata: Theory, Trends, And Applications*. Singapore: World Scientific Publishing Co Pte Ltd, 2023. ISBN 978-981-1278-12-9.

[24] Kozen, D. C. Two-Way Finite Automata. In: *Automata and Computability*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1977, p. 119–123. ISBN 978-3-642-85706-5.

[25] Kutrib, M.; Malcher, A. and Pighizzini, G. Oblivious two-way finite automata: Decidability and complexity. *Information and Computation*, 2014, vol. 237, p. 294–302. ISSN 0890-5401.

[26] Langholm, T. and Bezem, M. A Descriptive Characterisation of Even Linear Languages. *Grammars*, december 2003, vol. 6, p. 169–181. ISSN 1572-848X.

[27] Loukanova, R. Linear Context Free Languages. In: Jones, C. B.; Liu, Z. and Woodcock, J., ed. *Theoretical Aspects of Computing – ICTAC 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 351–365. ISBN 978-3-540-75292-9.

[28] McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 1943, vol. 5, no. 4, p. 115–133. ISSN 1522-9602.

[29] Meduna, A. *Automata and Languages: Theory and Applications*. London: Springer-Verlag, 2000. ISBN 978-1-85233-074-3.

[30] Meduna, A. *Formal Languages and Computation*. New York: Taylor & Francis Informa plc, 2014. Taylor and Francis. ISBN 978-1-4665-1345-7.

[31] Meduna, A. and Křivka, Z. *Jumping Computation: Updating Automata and Grammars for Discontinuous Information Processing*. Boca Raton: CRC Press LLC, 2024. ISBN 978-0-367-62093-6.

[32] Meduna, A. and Zemek, P. *Regulated Grammars and Automata*. New York: Springer US, 2014. ISBN 978-1-4939-0368-9.

[33] Mereghetti, C. and Pighizzini, G. Two-Way Automata Simulations and Unary Languages. *Journal of Automata, Languages and Combinatorics*, january 2000, vol. 5, no. 3, p. 287–300. ISSN 2567-3785.

[34] Parr, T. *The Definitive ANTLR 4 Reference*. 2ndth ed. Pragmatic Bookshelf, 2013. ISBN 978-1-934356-99-9.

[35] Petrov, S. and Okhotin, A. On the transformation of two-way finite automata to unambiguous finite automata. *Information and Computation*, december 2023, vol. 295, p. 1–26. ISSN 0890-5401. Article 104956.

[36] Pighizzini, G. Two-Way Finite Automata: Old and Recent Results. *Electronic Proceedings in Theoretical Computer Science*. Open Publishing Association, august 2012, vol. 90, p. 3–20. ISSN 2075-2180.

[37] Păun, G.; Rozenberg, G. and Salomaa, A. *DNA Computing: New Computing Paradigms*. Springer Berlin Heidelberg, 1998. Texts in Theoretical Computer Science. An EATCS Series. ISBN 978-3-540-64196-4.

[38] Qiu, D. Some Observations on Two-Way Finite Automata with Quantum and Classical States. In: Huang, D.-S.; Wunsch, D. C.; Levine, D. S. and Jo, K.-H., ed. *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 1–8. ISBN 978-3-540-87442-3.

[39] Rabin, M. O. and Scott, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, april 1959, vol. 3, no. 2, p. 114–125. ISSN 0018-8646.

[40] Ravikumar, B. On some variations of two-way probabilistic finite automata models. *Theoretical Computer Science*, 2007, vol. 376, no. 1, p. 127–136. ISSN 0304-3975.

[41] Sakoda, W. J. and Sipser, M. Nondeterminism and the size of two way finite automata. In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. New York: Association for Computing Machinery, 1978, p. 275–286. STOC '78. ISBN 978-1-4503-7437-8.

[42] Salomaa, A. *Formal Languages*. New York: Academic Press, 1973. ACM monograph series. ISBN 978-0-12-615750-5.

[43] Shepherdson, J. C. The Reduction of Two-Way Automata to One-Way Automata. *IBM Journal of Research and Development*, 1959, vol. 3, no. 2, p. 198–200. ISSN 0018-8646.

[44] Wood, D. *Theory of Computation*. Harper & Row, 1987. Harper & Row computer science and technology series. ISBN 978-0-06-047208-5.

[45] Yakaryilmaz, A. and Say, A. C. C. Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics & Theoretical Computer Science*, january 2010, vol. 12, no. 4, p. 19–40. ISSN 1365-8050.

[46] Zheng, S.; Li, L. and Qiu, D. Two-Tape Finite Automata with Quantum and Classical States. *International Journal of Theoretical Physics*, april 2011, vol. 50, no. 4, p. 1262–1281. ISSN 1572-9575.

[47] Zheng, S.; Qiu, D. and Li, L. Some Languages Recognized by Two-Way Finite Automata with Quantum and Classical States. *International Journal of Foundations of Computer Science*, 2012, vol. 23, no. 5, p. 1117–1129. ISSN 1793-6373.

Appendix A

Input-Erasing Two-Way Finite Automaton Simulator

In this appendix, we describe the implemented tool that simulates IE2GFAs (see Definition 4.3). This tool can simulate any IE2GFA under alternating computation, even computation, and initialized even computation (see Definition 6.1), as well as without any computational restrictions. First, we give a brief overview of the technologies chosen for its implementation. Then, we describe the implementation itself, including the design of this tool, and provide a comprehensive usage manual. Finally, we test the tool and discuss some of its properties and future improvements.

Technology

Since the simulator is intended primarily for demonstration purposes, it is implemented in Python 3.13[1] for its ease of use, extensive standard library, and broad support. These factors compensate for the potentially slower execution speed of the resulting program, caused mainly by the interpreted nature and high level of abstraction of this programming language.

Furthermore, for lexical and syntax analysis of input IE2GFA specifications, the simulator uses a parser generated by ANTLR v4[2] (ANother Tool for Language Recognition) [34]. This tool can generate any parser using only a grammar specification that defines its desired behavior. It can also be easily used with Python by means of the packages `antlr4-tools`[3] and `antlr4-python3-runtime`[4].

Design and Implementation

The simulator is implemented as a console application. As its input, it expects a file containing the specification of an IE2GFA, and strings on which it should simulate this automaton. Simulation results are then printed either to the standard output or to a specified file, depending on its input arguments.

[1] The documentation for Python 3.13 is available at https://docs.python.org/3.13/.
[2] ANTLR v4 is available at https://www.antlr.org.
[3] `antlr4-tools` is available at https://pypi.org/project/antlr4-tools/.
[4] `antlr4-python3-runtime` is available at https://pypi.org/project/antlr4-python3-runtime/.

The source code of the simulator is divided into several modules (files). The entry point of the program is the `main` function in the `main.py` module. The `args.py` module provides the `parse_cmd_args` function that processes the input arguments of the program, which are described below in this appendix. It is implemented using `ArgumentParser` from the `argparse` module, which belongs to the standard library. Next, the `file_helpers.py` module provides helper functions for file handling, the `output_helpers.py` module offers helper functions for formatting simulation results, and the `shared.py` module contains constants shared by several other modules. The core of the simulator, however, consists of the `ie2gfa_listener_interp.py` and `ie2gfa` modules. The former is responsible for parsing input IE2GFAs, while the latter simulates them. We describe these two parts of the simulator in more detail.

Parsing Input IE2GFA Specification

As mentioned above, the simulator performs lexical and syntax analysis of input IE2GFA specifications using a parser generated by ANTLR. The source grammar[5] describing the expected structure of these specifications, which is used to generate the parser, is located in the `ie2gfa.g4` file. The parser is listener-based[6]. First, it builds a parse tree of an input IE2GFA (more precisely, of its specification) and checks for any lexical and syntax errors. Then, it traverses the parse tree. During this process, for each internal parse tree node, the parser calls the `enter` and `exit` methods corresponding to the nonterminal that the node represents. The `enter` method is called the first time the node is encountered, and the `exit` method is called after all its children have been processed. ANTLR provides all these `enter` and `exit` methods with empty implementations in the automatically generated `ie2gfaListener` class. However, it is possible to override them in a subclass of this class. Therefore, the `IE2GFAListenerInterp` class is implemented. This subclass of the `ie2gfaListener` class, located in the `ie2gfa_listener_interp.py` module, checks for semantic errors in the input IE2GFA and prepares its internal representation used by the simulator.

IE2GFA Simulation

Internally, the simulator represents and simulates every input IE2GFA using the `IE2GFA` class located in the `ie2gfa.py` module. Although the states of the input IE2GFA are denoted by nonempty strings in its specification (the structure of which is described later in this appendix), the simulator maps them to nonnegative integers and works with them in this form. Since the input IE2GFA can be any IE2GFA, even nondeterministic, its simulation process is based on searching its configuration space in a depth-first manner. Thus, if the input IE2GFA can make several different moves from its current configuration, the simulator simulates one of them and later backtracks to continue with another one.

In general, the simulation process of an input IE2GFA on an input string is as follows. The simulator selects one of the start configurations of the IE2GFA for the given string and starts applying the rules of this automaton to it. This way, the simulator either finds a computation through which the IE2GFA accepts the string or determines that no such computation exists from the selected start configuration. If the string is accepted,

[5]The general ANTLR grammar form is described in Chapter 15 in [34] or at https://github.com/antlr/antlr4/blob/4.13.2/doc/grammars.md.

[6]ANTLR parse-tree listeners are described in Section 2.5 in [34] or at https://github.com/antlr/antlr4/blob/4.13.2/doc/listeners.md.

the simulation ends; otherwise, the simulator repeats the same process with one of the remaining start configurations. If no such configuration remains, the simulated IE2GFA rejects the string.

Although each simulation is performed as described above, certain IE2GFA moves can be simulated simultaneously. These are the moves that read the same symbol sequence in the same direction. Therefore, at each point during the simulation, instead of a single current state, the simulator maintains a set of states in which the simulated IE2GFA can currently be. This allows it to perform certain IE2GFA computations simultaneously and thus speed up the simulation process.

In its default setting, the simulator cannot simulate IE2GFAs that contain cycles consisting of ε-rules (in their diagrams), as it could cycle in them indefinitely. However, it can be run in a mode that avoids re-exploring already explored IE2GFA configurations. This allows it to simulate IE2GFAs with ε-rule cycles and also speed up simulations of some other IE2GFAs at the cost of a significant increase in memory requirements for simulations.

Note that all simulations of restricted (alternating, even, and initialized even) and unrestricted IE2GFA computations are performed in the same way. The only difference lies in the strategy of selecting moves for simulation.

Usage

This section provides instructions for installing and running the simulator.

Instalation

In order to use the simulator, all its dependencies must first be resolved. For convenience, it is highly recommended to use a Python virtual environment, which can be created using the `venv` module[7]. To create the virtual environment with all necessary packages installed, run the command `make venv`. Alternatively, use the following two commands:

```
python3 -m venv .ie2gfa_sim-venv
.ie2gfa_sim-venv/bin/pip install -r requirements.txt
```

Next, to generate the parser for input IE2GFA specifications from its grammar specification in the `ie2gfa.g4` file using ANTLR, run the command `make parser`. Alternatively, use the command

```
.ie2gfa_sim-venv/bin/antlr4 -Dlanguage=Python3 -o src/antlr4parser ie2gfa.g4
```

Finally, before running the simulator, activate the virtual environment using the command

```
source .ie2gfa_sim-venv/bin/activate
```

To resolve the dependencies and generate the parser without using the virtual environment, run the following two commands:

```
pip install -r requirements.txt
antlr4 -Dlanguage=Python3 -o src/antlr4parser ie2gfa.g4
```

[7] The documentation for the `venv` module is available at https://docs.python.org/3/library/venv.html.

Running

The simulator can be run using the following command:

```
./ie2gfa_sim [-h] [-d DATA_FILE] [-o OUTPUT_FILE] [-u|-a|-e|-i] [-c] [-p] SPEC_FILE
```

The only mandatory positional argument, `SPEC_FILE`, represents the file containing the specification of an IE2GFA that is to be simulated. The remaining arguments are described as follows:

(1) `-h`, `--help`: Print the help message to the standard output.

(2) `-d`, `--data_file DATA_FILE`: The file containing strings on which the IE2GFA is to be simulated. Each string should be on a separate line. All leading and trailing spaces on each line are ignored. Any blank line represents ε. If this file is not specified, these strings are read from the standard input.

(3) `-o`, `--output_file OUTPUT_FILE`: The file for storing the simulation results. If not specified, the results are printed to the standard output.

(4) `-u`, `--unrestricted`: Simulate the IE2GFA without any computational restrictions (the default simulation mode).

(5) `-a`, `--alternating`: Simulate the IE2GFA under alternating computation.

(6) `-e`, `--even`: Simulate the IE2GFA under even computation.

(7) `-i`, `--initialized_even`: Simulate the IE2GFA under initialized even computation.

(8) `-c`, `--computation_details`: For each accepted input string, extend the simulation results with the sequence of rules used to accept it.

(9) `-p`, `--prune`: During the simulation of the IE2GFA on each input string, avoid re-exploring already explored IE2GFA configurations. This can speed up simulations of nondeterministic IE2GFAs, whose nondeterministic choices lead to different positions of the read head on the input tape, and it also allows the simulator to simulate IE2GFAs with ε-rule cycles. However, it comes with a significant increase in memory requirements for simulations.

Note that the options `-u`, `-a`, `-e`, and `-i` are mutually exclusive, so no two of them can be set simultaneously. The simulator uses UTF-8 encoding to work with files.

Return Codes

After a successful run, the simulator prints the simulation results to the standard output or to the specified output file and returns 0. However, if the simulator fails to run correctly, an error message is printed to the standard error, and one of the following error codes is returned:

- 2: Invalid input arguments.
- 11: Syntax error in the input IE2GFA specification.
- 12: Semantic error in the input IE2GFA specification (occurrence of a state that is not listed in the set of states or of a symbol that is not in the input alphabet).
- 99: Internal error of the simulator (for example, when the simulator cannot open the specification file, the data file, or the output file).

IE2GFA Specification Format

The simulator supports two formats of the input IE2GFA specification—complete and short. The former is based on the structure of the formal description of IE2GFAs (see Definition 4.3) and follows these rules:

- Each line comment must be prefixed with `//`. The text following this prefix is then ignored until the end of the line. Similarly, each block comment must start with `/*` and end with `*/`. The text between these two markers is then ignored.
- Each symbol in the input alphabet of an input IE2GFA must belong to the set of symbols defined by the regular expression `[!-'*+.-;=?-z|~]`. Similarly, each state of an input IE2GFA must be a nonempty string over that same set of symbols.
- In each IE2GFA rule, the state and the read symbol sequence on the left-hand side of the rule (if the symbol sequence is nonempty) must be separated by `<` if the rule is left and by `>` if the rule is right. If the rule does not read any symbols, neither `<` nor `>` should be used on its left-hand side. The left-hand side and the right-hand side of the rule must be separated by `->`.
- All whitespace characters are ignored; however, they must never split `->` in any IE2GFA rule.

The short IE2GFA specification format is defined similarly to the complete one, except that it does not require the explicit specification of the set of states and the input alphabet. Examples of both of these IE2GFA specification formats are given below.

Examples

Here, we provide two examples of the usage of the simulator.

Example 1

Consider the file `ie2gfa_spec_complete.txt` containing the following IE2GFA specification (in the complete IE2GFA specification format):

```
(   // L = {a^nb^nc^m | m, n >= 0}
    {s, q, f},          // the set of states
    {a, b, c},          // the input alphabet
    {
        s>b -> q,
        a<q -> s,
        s>c -> f,
        f>c -> f
    },                  // the set of rules
    s,                  // the start state
    {s, f}              // the set of final states
)
```

Further, consider the file `data_file1.txt` containing the following strings:

```
aaabbbccc
aaaaaaabbb
cccc
```

Running the simulator using the command

```
./ie2gfa_sim ie2gfa_spec_complete.txt -d data_file1.txt
```

produces the following results:

```
================================================================
Simulated IE2GFA: ie2gfa_spec_complete.txt
Computation type: unrestricted
================================================================
Simulation 1

String: aaabbbccc
Result: ACCEPTED
----------------------------------------------------------------
Simulation 2

String: aaaaaaabbb
Result: REJECTED
----------------------------------------------------------------
Simulation 3

String: cccc
Result: ACCEPTED
----------------------------------------------------------------
```

Example 2

Consider the file `ie2gfa_spec_short.txt` containing the following IE2GFA specification corresponding to the IE2GFA from Example 4.7 (in the short IE2GFA specification format):

```
(   // L = {a,b}^*{ab}{a, b}^2
    s,                  // the start state
    {
        s>a -> q1,
        q1>b -> q2,
        q2>a -> q3,
        q2>b -> q3,
        q3>a -> f,
        q3>b -> f,
        a<f -> f,
        b<f -> f
    },                  // the set of rules
    {f}                 // the set of final states
)
```

Additionally, consider the file `data_file2.txt` containing the following strings:

```
abab
abbabaa
ababbaab
```

Running the simulator with the command

```
./ie2gfa_sim ie2gfa_spec_short.txt -d data_file2.txt -c
```

results in the following:

```
======================================================================
Simulated IE2GFA: ie2gfa_spec_short.txt
Computation type: unrestricted
======================================================================
Simulation 1

String: abab
Result: ACCEPTED

Used sequence of rules:
1: s>a -> q1
2: q1>b -> q2
3: q2>a -> q3
4: q3>b -> f
----------------------------------------------------------------------
Simulation 2

String: abbabaa
Result: ACCEPTED

Used sequence of rules:
1: s>a -> q1
2: q1>b -> q2
3: q2>a -> q3
4: q3>a -> f
5: b<f -> f
6: b<f -> f
7: a<f -> f
----------------------------------------------------------------------
Simulation 3

String: ababbaab
Result: REJECTED
----------------------------------------------------------------------
```

Testing and Evaluation

The simulator was tested on the example inputs provided in the `examples` directory. However, even though it works correctly on them, testing confirmed the following problem

related to the simulation of certain nondeterministic IE2GFAs. Consider the ε-free IE2SFA M given in Figure A.1 below with $w = ba^nb^n$, for some $n \geq 0$, as its input string. Clearly, M rejects w. However, to reach this conclusion, the simulator must perform all possible computations of M on w. Since, starting from the configuration ba^nsb^n, M can make moves according to both of its rules from the majority of the configurations that it reaches, the number of such computations grows exponentially with n. However, as no two different moves from the same configuration can be simulated simultaneously due to their opposite directions, the simulator performs these computations, in essence, one at a time. Unfortunately, this becomes quite time-consuming, even for smaller values of n. This problem is partially solved by the option not to re-explore any already explored configurations. For this reason, however, the simulator must be able to store these configurations, which significantly increases its memory requirements. In the current implementation, this is done by recording sets of already explored states for all positions of the read head on the input tape reached during a simulation. Since the number of such positions is upper-bounded by the function $(m + 1) + m + \ldots + 1 = \frac{1}{2}(m + 2)(m + 1)$, where m represents the length of an input string, the number of such records can grow relatively quickly.

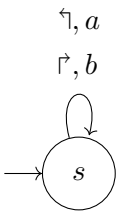

Figure A.1: State diagram of the ε-free IE2SFA $M = (\{s\}, \{a, b\}, \{as \to s, sb \to s\}, s, \{s\})$.

As mentioned earlier, the simulator works with sets of states of an input IE2GFA. Each such set encountered during a simulation is mapped to a unique integer value that represents it, so it is stored only once. However, to further reduce the memory requirements of a simulation, these sets could be represented more efficiently, for instance, by binary decision diagrams (see [7]). Furthermore, the simulator could also be extended in the future with a graphical user interface or the ability to simulate IE2GFA computations step by step.

Appendix B

Contents of the External Attachments

The external attachments are organized as follows:

```
/
├── ie2gfa-simulator/ ..... IE2GFA simulator implementation
│   ├── examples/ ..... Examples of input IE2GFA specifications and strings
│   ├── ie2gfa_sim ..... Script to run the simulator
│   ├── ie2gfa.g4 ..... Grammar for generating the IE2GFA specification parser for the simulator with ANTLR
│   ├── Makefile ..... To prepare the environment for running the simulator
│   ├── README.md ..... Running instructions and usage examples
│   ├── requirements.txt ..... Python dependencies of the simulator
│   └── src/ ..... Source files of the simulator
└── thesis-text/ ..... Directory related to the text of this thesis
    ├── src/ ..... LaTeX source files of this thesis
    ├── xnejed09-thesis.pdf ..... PDF version of this thesis
    └── xnejed09-thesis-print.pdf ..... PDF version of this thesis for printing
```