BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

ABSTRACTION IN AUTOMATA ALGORITHMS

ABSTRAKCE V AUTOMATOVÝCH ALGORITMECH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TOMÁŠ KOCOUREK**
AUTOR PRÁCE

SUPERVISOR **doc. Mgr. LUKÁŠ HOLÍK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2021/2022

Bachelor's Thesis Specification

Student: **Kocourek Tomáš**
Programme: Information Technology
Title: **Abstraction in Automata Algorithms**
Category: Algorithms and Data Structures
Assignment:
Finite automata are often combined using Boolean and other operators, which are prone to state space explosion and easily become an efficiency bottleneck. Alternating automata allow to circumvent this problem by allowing a succinct implicit representation of Boolean operations. The price for this is an expensive language emptiness test. The goal of this work is to investigate whether algorithm from [1], which uses abstraction, can significantly reduce the cost of alternating automata emptiness testing.

1. Familiarise yourself with alternating automata and with the algorithm for testing language emptiness of alternating automata from [1].
2. Implement the algorithm from [1].
3. Compare the performance of your implementation at least with the antichain based alternating automata emptiness test, if possible also with other algorithms.

Recommended literature:

- Ganty P., Maquet N., Raskin JF. (2009) Fixpoint Guided Abstraction Refinement for Alternating Automata. In: Implementation and Application of Automata. CIAA 2009. Lecture Notes in Computer Science, vol 5642. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02979-0_19
- Loris D'Antoni, Zachary Kincaid, Fang Wang, A Symbolic Decision Procedure for Symbolic Alternating Finite Automata, Electronic Notes in Theoretical Computer Science, Volume 336, 2018, Pages 79-99, ISSN 1571-0661, https://doi.org/10.1016/j.entcs.2018.03.017.

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

Supervisor: **Holík Lukáš, doc. Mgr., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: November 3, 2021

Abstract

The goal of this thesis is to implement and experimentally compare antichain-based algorithms with and without abstraction, which decide the emptiness of alternating finite automata. The author also proposes his own algorithms using abstraction and comes up with a few optimizations of existing abstract algorithms. The thesis introduces the theoretical background of studied algorithms and describes efficient ways to implement data structures which are used by these algorithms. The experimental evaluation over random automata shows that the algorithms without abstraction give us better results in general because they do not perform costly evaluation of closed set intersection and complementation. However, in case of automata with high transition density, the algorithms without abstraction tend to decelerate, while the abstract ones accelerate.

Abstrakt

Tato práce si klade za cíl implementaci a experimentální porovnání protiřetězcových algoritmů s abstrakcí a bez abstrakce, které testují prázdnost alternujících automatů. Autor také navrhuje vlastní algoritmy s abstrakcí a navrhuje několik optimalizací pro existující abstraktní algoritmy. Práce popisuje teoretické pozadí studovaných algoritmů a navrhuje efektivní způsob implementace datových struktur, které jsou těmito algoritmy používány. Experimentální vyhodnocení na náhodných automatech ukazuje, že algoritmy bez abstrakce vykazují obecně lepší výsledky, neboť nevyužívají náročné operace průniku a komplementace shora a zdola uzavřených množin. V případě automatů s vysokou hustotou přechodů však algoritmy bez abstrakce zpomalují a algoritmy s abstrakcí naopak zrychlují.

Keywords

alternating finite automaton, abstraction, antichain, language emptiness, concrete domain, abstract domain, fixed point, partition

Klíčová slova

alternující konečný automat, abstrakce, protiřetězec, prázdnost jazyka, konkrétní doména, abstraktní doména, pevný bod, rozklad

Reference

KOCOUREK, Tomáš. *Abstraction in Automata Algorithms*. Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Mgr. Lukáš Holík, Ph.D.

Rozšířený abstrakt

Tato práce si klade za cíl prozkoumat existující řešení pro testování prázdnosti alternujících automatů pomocí protiřetězcových algoritmů s abstrakcí a bez abstrakce, implementovat tyto algoritmy a experimentálně vyhodnotit jejich efektivitu s pomocí alternujících automatů generovaných náhodně dle rozličných parametrů.

Práce uvádí základní pojmy, jako je nedeterministický konečný automat (*NFA*), alternující konečný automat (*AFA*) a objasňuje, že naivní převod *AFA* na *NFA* s následným provedením všeobecně známého testu prázdnosti tohoto *NFA* může vést k exponenciálnímu nárustu stavů, což nás motivuje k nalezení efektivnějšího řešení pro testování prázdnosti *AFA*.

V práci jsou podrobně představeny protiřetězcové algoritmy pro testování prázdnosti *AFA*, které pracují s abstrakcí (abstraktní algoritmy), a které pracují bez ní (konkrétní algoritmy). Autor dále navrhuje vlastní abstraktní obousměrné algoritmy, které během svého běhu počítají více užitečných informací, jež mohou vést k rychlejšímu nalezení řešení.

Následně jsou představeny datové struktury pro alternující automaty, inverzní přechodové relace, shora a zdola uzavřené množiny a rozklady množin, jež jsou koncipovány tak, aby zvýšily efektivitu provádění dílčích operací v rámci studovaných algoritmů.

V rámci experimentálního vyhodnocení implementovaných algoritmů práce nejprve představuje model Tabakova-Vardiho, jenž slouží k náhodnému generování *NFA* dle zvolené mohutnosti množiny stavů, hustoty přechodové funkce a poměru koncových stavů ke všem stavům. Autor následně představuje vlastní model generátoru automatů, jenž původní model Tabakova-Vardiho rozšiřuje a umožňuje generovat také *AFA* pomocí parametru míry alternace. Přináší rovněž možnost pomocí parametru opětovného navštěvování stavů lépe kontrolovat strukturu generovaných automatů a potlačit pravděpodobnost, že náhodně vygenerovaný automat bude sestávat z mnoha malých, vzájemně separovaných komponent.

Úvodní pozorování výsledků experimentálního vyhodnocení studovaných algoritmů nad 12 000 *AFA* ukazuje, že abstraktní algoritmy v průměru pracují výrazně méně efektivně než algoritmy konkrétní. V práci jsou proto poté zkoumány důvody tohoto významného rozdílu prostřednictvím měření času vykonávání jednotlivých operací v rámci abstraktních algoritmů.

Měření ukazuje, že průnik a komplementace shora a zdola uzavřených množin představují významně náročné operace v porovnání s ostatními procedurami. Je proto navrženo několik optimalizací, které modifikují původní abstraktní algoritmy i nově vymyšlené abstraktní obousměrné algoritmy tak, aby nebylo nutné tyto náročné výpočty provádět tak často. Jedna z modifikací, která výměnou za snížení přesnosti informací počítaných v rámci algoritmu úplně odstraňuje nutnost provádění průniku shora a zdola uzavřených množin mimo abstraktní doménu, přináší tak výrazné snížení průměrného času provádění abstraktních algoritmů, že při zvyšující se mohutnosti množiny stavů vykazuje stále lepší výsledky než jeden z konkrétních algoritmů.

Následně je popsáno, jak lze problémy inkluze a průniku *NFA* řešit pomocí testu prázdnosti alternujícího automatu. Několik stovek instancí těchto problémů je následně vygenerováno a vyhodnoceno pomocí studovaných algoritmů. Ukazuje se, že v těchto případech vykazují abstraktní dopředné algoritmy lepší výsledky než konkrétní dopředný algoritmus, zvyšujeme-li hustotu přechodů jednotlivých komponent průniku nebo inkluze.

Abstraction in Automata Algorithms

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Mgr. Lukáš Holík, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Tomáš Kocourek
May 18, 2022

Acknowledgements

I would like to thank to doc. Mgr. Lukáš Holík, Ph.D. for his priceless advice, professional guidence and also for his patience.

Contents

1 Introduction **3**

2 Preliminaries **5**
2.1 Nondeterministic Finite Automaton . . . 6
2.2 Language Emptiness Test of an NFA . . . 7
2.3 Alternating Finite Automaton . . . 8
2.4 AFA to NFA Transformation . . . 9
2.5 Naive Language Emptiness Test of an AFA . . . 10

3 Concrete Forward and Backward Algorithms Deciding the AFA Emptiness **11**
3.1 Predicate Transformers . . . 11
3.2 Fixed Points of Predicate Transformers . . . 14
3.3 Antichains . . . 15
3.3.1 Significant Properties of Antichains . . . 16
3.4 AFA Emptiness . . . 18

4 Abstract Forward and Backward Algorithms Deciding the AFA Emptiness **20**
4.1 Lattice of Partitions . . . 20
4.2 Concrete and Abstract Domain . . . 22
4.2.1 Abstraction and Concretization Functions . . . 22
4.2.2 Abstraction and Concretization Functions over Formulae . . . 23
4.3 Abstract Alternating Automaton . . . 24
4.3.1 Language of an Abstract AFA . . . 25
4.3.2 Predicate Transformers over Abstract AFA . . . 27
4.4 Representability of Concrete Nodes in an Abstract Domain . . . 27
4.5 The Abstract Algorithms . . . 28
4.5.1 The Abstract Forward Algorithm . . . 28
4.5.2 The Abstract Backward Algorithm . . . 30

5 Abstract Bidirectional Algorithms Deciding the AFA Emptiness **31**
5.1 Motivation of Using a Bidirectional Approach . . . 31
5.2 Forward-like Abstract Bidirectional Algorithm . . . 32
5.3 Backward-like Abstract Bidirectional Algorithm . . . 36
5.4 Properties of Forward-like and Backward-like Algorithms . . . 36

6 Design and Implementation **37**

6.1 Alternating Automata 37
6.2 Closed Sets 38
6.3 Predicate Transformers 39
6.4 Partitions 41

7 Experimental Comparison of Algorithms Deciding AFA Emptiness 42
7.1 Tabakov-Vardi Model 42
7.2 Extended Tabakov-Vardi Model 43
7.3 Initial Observation of Randomly Generated AFAs Behaviour 44
7.4 Comparison of Studied Algorithms in Context of Emptiness of NFA Intersection 49
7.5 Comparison of Studied Algorithms in Context of NFA Inclusion 51

8 Conclusion 54

Bibliography 56

Chapter 1

Introduction

A *finite automaton* (*FA*) is a well-known formal computation model with many various applications across the computer science.

An *alternating finite automaton* (*AFA*), which represents a generalization of the *nondeterministic FA* (*NFA*) and the *parallel machine*, is a concept with precisely the same computing power as *FA*.

Nevertheless, it allows us to express the same model in a more compact way using Boolean formulae. Thus, an *AFA* is not predisposed to operate with a massive amount of states.

The price for this is the necessity to cope with the costly language emptiness test because it is not possible to efficiently utilize the straightforward language emptiness test commonly used for *FA*.

The naive method to check the language emptiness requires transforming an *AFA* to an equivalent *NFA* and then using the conventional algorithm for language emptiness. Unfortunately, this procedure would explosively increase the number of states of the automaton. In the worst case, the amount of states expands exponentially, which we want to avoid.

Therefore, it is essential to study alternative techniques for effective language emptiness test of an *AFA*.

First, we present two *antichain-based* algorithms, which work in a forward and in a backward fashion. Next, this thesis introduces remarkable *abstraction refinement algorithms* by *Ganty*, *Maquet* and *Raskin* [4] based on conversion between concrete and abstract domains of *AFA*s and antichains utilisation. The implementation of the mentioned algorithm will be described, and it will be experimentally compared with the alternative algorithms which deal with the same problem.

In addition, we present two new proposed algorithms, which were created by the author of this thesis and which are derived from the studied abstraction refinement algorithms, we discuss the differences between them and we also experimentally compare their performance over plenty of various *AFA*s.

Organization of the Thesis

Firstly, we present the basic theoretical background within Chapter 2. Both concepts of *NFA* and *AFA* will be described in detail and the idea of the naive language emptiness test of an *AFA* will be briefly discussed.

Subsequently, Chapter 3 introduces the idea of predicate transformers, which helps us to inspect the behaviour of an *AFA*, and fixed point expressions over them and clarify the relation between reachability of an *AFA* and fixed points of predicate transformers. The theory of antichains, which comprises the core of language emptiness test presented at the end of Chapter 3, will be also discussed.

While the language emptiness test shown in Chapter 3 operates in the concrete domain of an *AFA*, which means that it does not use any kind of reduction of its state set, Chapter 4 finally brings the concept of abstraction of an *AFA*. This technique uses a partition of a state set to create a smaller automaton, which over-approximates the language of the original one. The possibility of using abstract automata to decide the emptiness of the concrete one will be described in detail.

Within Chapter 5, we will present new proposed abstract bidirectional algorithms deciding an *AFA* emptiness, which were made up by the author of this thesis. While the former abstract algorithms introduced in Chapter 4 work exclusively in the forward or backward fashion, we discuss the possibility of discover more information about an automaton at once using a bidirectional approach.

Next, Chapter 6 discusses the design of data structures proposed by author of this thesis, which allows us to efficiently represent necessary information used to decide an *AFA* emptiness by presented algorithms. This chapter also deals with implementation details.

Afterwards, Chapter 7 describes experimental evaluation of introduced algorithms. The chapter brings the concept of *Tabakov-Vardi* model of randomly generated *NFA*s, which is subsequently extended by the author of this thesis to the model which allows us to generate *AFA*s using various parameters. These randomly generated automata will be used to perform experiments. The results of these experiments will be also presented within the chapter.

Chapter 2

Preliminaries

This section introduces the essential theoretical background for the thesis. First, it presents formal terms like an *alphabet*, *language*, *nondeterministic finite automaton* and others connected with finite automata, which were inspired by [7]. The term *positive Boolean formula*, *alternating finite automaton* and corresponding concepts, which were all inspired by [4], will be also discussed. Both nondeterministic and alternating finite automata are defined with a single initial state within the referred literature. Since we decided to define these models more generally, we allow an existence of multiple initial states, which means that we had to adjust other definitions to this modification as well.

Definition 1 *An* ***alphabet*** Σ *is defined as a finite, non-empty set of symbols.*

Definition 2 *The finite sequence of symbols* $w = a_0 \ldots a_n$*, where* $\forall i \in \mathbb{N} : i \leq n \Rightarrow a_i \in \Sigma$*, is called a* ***string*** *over an alphabet* Σ*.* $|w| = n$ *is a length of the string. Specifically,* $\epsilon \notin \Sigma$ *is denoted as an* ***empty string****, which has the unique property that* $|\epsilon| = 0$*.*

Definition 3 Σ^* *represents an infinite set of all possible strings over an alphabet* Σ*. Each subset* $L \subseteq \Sigma^*$ *is defined as a* ***language*** *over an alphabet* Σ*.*

Definition 4 *Let* A *be a set. A* ***positive Boolean formula*** φ *over a set* A *is defined as the formula which takes the following form* $\varphi \triangleq a \mid \bot \mid \top \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$*, for* $a \in A$*. The set* $B^+(A)$ *represents the union of all possible positive Boolean formulae over the set* A*.*

Suppose that A is a set of propositions. We denote by $[\![.]\!]$ a denotation function $[\![.]\!] : B^+(A) \rightarrow \mathcal{P}(\mathcal{P}(A))$, which maps a positive Boolean formula φ to a set $[\![\varphi]\!]$, whose elements represent all the possible valuations that satisfy the formula φ. For an individual valuation $V \in [\![\varphi]\!]$, the proposition $v \in V$ is assumed to be true, whereas each proposition $v \in A \setminus V$ is perceived as false. Consider $\phi, \xi \in B^+(A)$ to be two formulae. We define $[\![\phi \vee \xi]\!] = [\![\phi]\!] \cup [\![\phi]\!]$ and analogously $[\![\phi \wedge \xi]\!] = [\![\phi]\!] \cap [\![\xi]\!]$.

Example. Let $A = \{a, b\}$, $\varphi_A \in B^+(A)$, since $\varphi_A = (a \wedge b) \vee b$. Then $[\![\varphi_A]\!] = \{\{b\}, \{a, b\}\}$ because both $\{b\}$ and $\{a, b\}$ are sets which satisfy the formula φ_A if their elements are considered to be true.

2.1 Nondeterministic Finite Automaton

Definition 5 *Formally, a **nondeterministic finite automaton** A (shortly denoted as* NFA *A) is a quintuple $A = (Q, \Sigma, S_0, R, F)$, where*

- *Q is a finite set of states*
- *Σ is an alphabet, $\Sigma \cap Q = \varnothing$*
- *$S_0 \subseteq Q$, where $S_0 \neq \varnothing$, is a set of initial states*
- *R is a transition relation $R \subseteq Q \times \Sigma \times Q$, which represents a finite set of transition rules*
- *$F \subseteq Q$ represents a finite set of final states*

Nondeterministic finite automaton (or alternatively known as *nondeterministic finite state machine*) is a formal mathematical model of computation representing a *regular language* [7].

Definition 6 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA*. A tuple $\tau = (q, w)$, where $q \in Q$ and $w \in \Sigma^*$, is called a **configuration** of an automaton A.*

Definition 7 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA*. Suppose that $(q_0, a_0, q_1) \in R$ is a transition rule such that $q_0, q_1 \in Q, a_0 \in \Sigma$ and let both $\tau_0 = (q_0, a_0 w), \tau_1 = (q_1, w)$ be configurations of A, where $w \in \Sigma^*$.*

*Then, the **transition** from the state q_0 to the state q_1 using the character a_0 in accordance to the rule (q_0, a_0, q_1) is symbolically denoted as $\tau_0 \vdash \tau_1$.*

Definition 8 *Let $A = (Q, \Sigma, S_0, R, F)$ be an NFA, $\tau_0, \tau_1, ...\tau_n$ are configurations of A. Assume that there exist appropriate transition rules from R such that it is possible to perform each transition $\tau_i \vdash \tau_{i+1}$ where $i \in \mathbb{N} \wedge i < n$.*

*In this case, a sequence $\tau_0 \vdash \tau_1 \vdash \ldots \vdash \tau_n$ is called a **run** of A according to rules from R. We can simply denote this fact as $\tau_0 \vdash^* \tau_n$ using the reflexive and transitive closure of $\vdash$.*

Definition 9 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA*. We say that the string $w \in \Sigma^*$ is **accepted** by the automaton A if and only if there exists a run $\tau_0 \vdash^* \tau_n$ of A for some $n \in \mathbb{N}$, where $\tau_0 = (s_0, w)$, $\tau_n = (f, \epsilon), f \in F$ and $s_0 \in S_0$.*

*Otherwise, the string w is **not accepted** by the automaton A.*

Definition 10 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA*. The **accepted language** $L(A)$ of the automaton A is equivalent to the set of all accepted strings of A. The automaton A **accepts** the language $L(A)$.*

Specifically, if the equation $L(A) = \varnothing$ holds, the automaton A does not accept any single string. We say that the automaton represents an *empty language*. This fact could be determined by a procedure called *language emptiness test*.

2.2 Language Emptiness Test of an NFA

Since every *NFA* could be understood as a directed graph, we can use an algorithmic way of searching for an accepting run of the given automaton. In what follows, two simple algorithms which use the idea of *terminating* and *reachable* sets of an *NFA* will be introduced.

Definition 11 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA. *We say that the state $q \in Q$ is **terminating** if there exists a run $\tau_0 \vdash^* \tau_n$ such that $\tau_0 = (q, w), \tau_n = (f, \epsilon)$, where $w \in \Sigma^*$ and $f \in F$.*

An intuitive language emptiness test is shown in Algorithm [1]. The algorithm operates with an idea of terminating states of an *NFA*. It tries to decide whether at least one of the initial states $s_0 \in S_0$ could be considered to be terminating by iteratively building a complete set of terminating states of the given *NFA*.

It is easy to see that the language of the given *NFA* is empty if and only if none of the initial states $s_0 \in S_0$ is terminating.

Algorithm 1: Language emptiness test of an NFA

Input: NFA $A = (Q, \Sigma, S_0, R, F)$
Output: True iff $L(A) = \varnothing$

```
1 T_0 ← F;
2 i ← 0;
3 do
4 |   i ← i + 1;
5 |   T_i ← T_{i−1} ∪ {p | (p, a, s) ∈ R ∧ s ∈ T_{i−1}};
6 while T_i ≠ T_{i−1};
7 return S_0 ∩ T_i = ∅;
```

Definition 12 *Let $A = (Q, \Sigma, S_0, R, F)$ be an* NFA. *We say that the state $q \in Q$ is **reachable** if there exists a run $\tau_0 \vdash^* \tau_n$ such that $\tau_0 = (s_0, w_1 w_2), \tau_n = (q, w_2)$, where $w_1, w_2 \in \Sigma^*$ and $s_0 \in S_0$.*

Analogously, it is possible to iteratively compute a set of all reachable states of an AFA and check whether there exists any final state of the given AFA which is reachable. This idea is summarized in Alghoritm [2].

Algorithm 2: Language emptiness test of an NFA

Input: NFA $A = (Q, \Sigma, S_0, R, F)$
Output: True iff $L(A) = \varnothing$

```
1 I ← S_0;
2 i ← 0;
3 do
4 |   i ← i + 1;
5 |   I_i ← I_{i−1} ∪ {s | (p, a, s) ∈ R ∧ p ∈ I_{i−1}};
6 while I_i ≠ I_{i−1};
7 return F ∩ I_i = ∅;
```

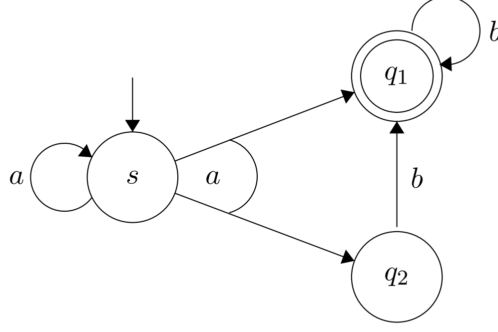

Figure 2.1: An example of an alternating finite automaton

2.3 Alternating Finite Automaton

Definition 13 ***Alternating finite automaton*** *A (shortly denoted as* AFA*) is a quintuple $A = (Q, \Sigma, S_0, \delta, F)$, where*

- *Q is a finite set of states*
- *Σ is an alphabet, $\Sigma \cap Q = \varnothing$*
- *$S_0 \subseteq Q$, where $S_0 \neq \varnothing$, is a set of initial states*
- *$\delta : Q \times \Sigma \to B^+(Q)$ stands for a discrete transition function, where $B^+(Q)$ is a set of positive Boolean formulae over Q*
- *$F \subseteq Q$ is a finite set of final states*

Example. Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA*, $Q = \{s, q_1, q_2\}$, $\Sigma = \{a, b\}$, $S_0 = \{s\}$, $\delta = \{((s, a), (q_1 \wedge q_2) \vee s),\ ((s, b), \bot), ((q_1, a), \bot), ((q_1, b), q_1), ((q_2, a), \bot), ((q_2, b), q_1)\}$, $F = \{q_1\}$.

The graphical representation of the automaton A is shown in Figure 2.1. Note that the arc which forms a circular sector around a symbol of Σ represents an universal transition, while the existential transition is symbolised by explicitly mentioning the character by each transition.

In future examples of *AFA*s, all tuples $(s, a) \subseteq Q \times \Sigma$ which are mapped to $\bot$ by a function δ will be left implicit.

Definition 14 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. In what follows, each set $N \subseteq Q$ will be referred as a **node** of the* AFA *A. In case of $N \cap S_0 \neq \varnothing$, we say that N is an **initial node** of A. A set of all initial nodes of A will be denoted by $I_0(A)$. In the other way, N is a **final node** as soon as $N \subseteq \mathcal{P}(F)$.*

Definition 15 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. Suppose that N is a node of A. A tuple $\tau = (N, w)$, where $w \in \Sigma^*$, is called a **configuration** of an* AFA *A.*

The essential difference between a configuration of an *AFA* and a configuration of an *NFA* lies in its first component. While an *NFA* is always in exactly one state, an *AFA* could be located in *multiple* states *in paralell* because of the universal transitions, which is expressed using the concept of nodes of an *AFA*.

Definition 16 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. Consider N_0, N_1 to be two nodes of A and $a \in \Sigma$. The triplet (N_0, a, N_1) is an **edge** of A if and only if $N_1 \in [\![\bigwedge_{q \in N_0} \delta(q, a)]\!]$. We will denote by $E(A)$ a **set of all edges** of A.*

Definition 17 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. Suppose that $\tau_0 = (N_0, aw), \tau_1 = (N_1, w)$ are two configurations of A and (N_0, a, N_1) is an edge of A. Under these conditions, it is possible to perform a **transition** from all states of N_0 in parallel using the symbol a. We will denote this fact as $\tau_0 \vdash_{E(A)} \tau_1$.*

Definition 18 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*, $\tau_0, \tau_1, \ldots, \tau_n$ are configurations of A. Assume that the discrete transition function δ allows the automaton to perform each transition $\tau_i \vdash_{E(A)} \tau_{i+1}$ where $i \in \mathbb{N} \land i < n$.*

*In this case, the sequence $\tau_0 \vdash_{E(A)} \tau_1 \vdash_{E(A)} \ldots \vdash_{E(A)} \tau_n$ is a **run** of the given* AFA*. Similarly to the definition of the sequence of transitions of an* NFA*, we can simply write $\tau_0 \vdash^*_{E(A)} \tau_n$ using the reflexive and transitive closure of $\vdash_{E(A)}$.*

Definition 19 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. The string $w \in \Sigma^*$ is **accepted** by the automaton A if and only if there exists a run $\tau_0 \vdash^* \tau_n$ of A for some $n \in \mathbb{N}$ such that $\tau_0 = (N_0, w)$ and $\tau_n = (N_f, \epsilon)$, where $N_0 \in I_0(A)$ is an initial node and $N_f \subseteq \mathcal{P}(F)$ is a final node.*

Definition 20 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. An **accepted language** $L(A)$ of the automaton A is the set of all accepted strings of A.*

Compared to the NFA, which accepts a string as soon as a configuration (f, ϵ), where $f \in F$ is reached, an AFA has to stop reading the input string in all the parallel branches all at once to accept a string.

Example. Let A be an AFA defined above in Example 2.3.1 and shown in Figure 2.1. Suppose that we work with the input string $w = ab$.

In view of the fact that there exists a run $(\{s\}, ab) \vdash (\{q_1, q_2\}, b) \vdash (\{q_1\}, \epsilon)$ and also $\{q_1\} \in \mathcal{P}(F)$, the string w is accepted by A.

2.4 AFA to NFA Transformation

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA* and $\varphi \in B^+(Q)$. Since δ maps its input to an element of $B^+(Q)$ and each valuation $N \in [\![\varphi]\!]$ is in fact a node of A, we can use the idea of such a subset construction to convert A to a corresponding *NFA* A_{NFA}, such that the language $L(A) = L(A_{NFA})$.

We define $Q_{NFA} = \mathcal{P}(Q), S_{NFA} = I_0(A), R = E(A)$ and $F_{NFA} = \mathcal{P}(F)$. Then, the quintuple $A_{NFA} = (Q_{NFA}, \Sigma, S_{NFA}, R, F_{NFA})$, forms such an *NFA* that the equation $L(A) = L(A_{NFA})$ holds. Therefore, the *AFA* A and the *NFA* A_{NFA} are equivalent.

This idea is summarized in the Algorithm 3 below.

Algorithm 3: *AFA* to *NFA* transformation

Input: AFA $A = (Q, \Sigma, S_0, \delta, F)$
Output: NFA $A_{NFA} = (Q_{NFA}, \Sigma, S_{NFA}, R, F_{NFA})$

1 $Q_{NFA} \leftarrow \mathcal{P}(Q)$;
2 $R \leftarrow \varnothing$;
3 $S_{NFA} \leftarrow \bigcup_{s_0 \in S_0} [\![s_0]\!]$;
4 **for** (p, a, s) **in** $Q_{NFA} \times \Sigma \times Q_{NFA}$ **do**
5   **if** $s \in \bigcap_{q \in p} [\![\delta(q, a)]\!]$ **then**
6         $R \leftarrow R \cup \{(p, a, s)\}$;
7 **end**
8 $F_{NFA} \leftarrow \mathcal{P}(F)$;
9 **return** $(Q_{NFA}, \Sigma, S_{NFA}, R, F_{NFA})$

2.5 Naive Language Emptiness Test of an AFA

The intuitive algorithm for checking language emptiness of an AFA utilizes all the ideas explained above.

Since it is possible to easily perform a language emptiness test over an NFA, the core of the naive test is based on transforming an AFA to an equivalent NFA using the subset construction introduced in Algorithm 3. Then, we compute either the set of all terminating states of the NFA, or the set of all reachable states of the NFA using the Algorithm 1 or Algorithm 2, respectively, to decide whether an original automaton is empty or not.

It is well known that for every finite set S, the cardinality of its power set $|\mathcal{P}(S)| = 2^{|S|}$. Due to the fact that the set of states of the *NFA* computed by the Algorithm 3 contains each node of the given *AFA*, we conclude that such a process leads to the exponential explosion of states.

Thus, the naive language emptiness test of an *AFA* is unfortunately not effective at all. In the following chapters, more sophisticated methods which omit the costly explicit *AFA* to *NFA* transformation, will be introduced.

Chapter 3

Concrete Forward and Backward Algorithms Deciding the AFA Emptiness

In was explained in Section 2.5 that the language emptiness test of an *AFA*, which uses explicit conversion from a given *AFA* to the corresponding *NFA*, is prone to state space explosion. Thus, it is required to avoid the conversion and run an emptiness test directly over a given *AFA*.

This chapter introduces the theoretical background for the *forward* and *backward* algorithms deciding the *AFA* emptiness in the *concrete* domain of the given *AFA* [4].

First, the concept of *predicate transformers*, which allow us to explore behaviour of the given *AFA* by inspecting its edges without explicitly computing the whole set $E(A)$, will be described [4].

Then, it will be shown how the concept of a *fixed point* [9, 6] of predicate transformers is related to a set of terminating and reachable sets of an *AFA* and it will be discussed how it is possible to compute such a fixed point.

The following sections introduces an idea of *antichains*, which significantly facilitate the evaluation of predicate transformers.

At the end of the chapter, both forward and backward algorithms for testing the *AFA* emptiness will be finally presented using the theory of preceding sections.

3.1 Predicate Transformers

Definition 21 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. Assume that there exists an edge $(N_0, a, N_1) \in E(A)$. In what follows, such a node N_1 will be called a **successor** of N_0. Dually, N_0 is a **predecessor** of N_1.*

The predicate transformer $post_A : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$ creates a set of all the successors of the nodes on its input considering the given *AFA* A. Formally, this fact can be expressed as follows:

$$post_A(X) = \bigcup_{a \in \Sigma} \{N_1 \mid \exists (N_0, a, N_1) \in E(A) : N_0 \in X\}.$$

Similarly, the predicate transformer $pre_A : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$, constructs a set of all the predecessors of its input according to the given AFA A, which is summarized below:

$$pre_A(X) = \bigcup_{a \in \Sigma} \{N_0 \mid \exists (N_0, a, N_1) \in E(A) : N_1 \in X\}.$$

Definition 22 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA, X is a set of nodes of A and N is a single node of A. Suppose that $\forall (N_0, a, N) \in E(A) : N_0 \in X$. Then, the node N is called a* ***controlled successor*** *of X. Analogously, the node N is called a* ***controlled predecessor*** *of X if the proposition $\forall (N, a, N_0) \in E(A) : N_0 \in X$ is true.*

The following equation describes the predicate transformer $\widetilde{post}_A : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$, which takes a set of nodes $X \subseteq \mathcal{P}(Q)$ as its input and transforms it to the set of all the controlled successors of nodes in X.

$$\widetilde{post}_A(X) = \bigcap_{a \in \Sigma} \{N_1 \mid \forall (N_0, a, N_1) \in E(A) : N_0 \in X\}.$$

Likewise, the predicate transformer $\widetilde{pre}_A : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$ maps its argument $X \subseteq \mathcal{P}(Q)$ to a set of all the controlled predecessors of elements in X. This idea is summarized in the following equation:

$$\widetilde{pre}_A(X) = \bigcap_{a \in \Sigma} \{N_0 \mid \forall (N_0, a, N_1) \in E(A) : N_1 \in X\}.$$

Note that since for all $X_0, X_1 \subseteq \mathcal{P}(Q)$ holds that

$$\begin{aligned}
& post_A(X_0 \cup X_1) \\
= & \bigcup_{a \in \Sigma} \{N_1 \mid \exists (N_0, a, N_1) \in E(A) : N_0 \in X_0 \cup X_1\} \\
= & \bigcup_{a \in \Sigma} \Big(\{N_1 \mid \exists (N_0, a, N_1) \in E(A) : N_0 \in X_0\} \cup \{N_1 \mid \exists (N_0, a, N_1) \in E(A) : N_0 \in X_1\} \Big) \\
= & \, post_A(X_0) \cup post_A(X_1),
\end{aligned}$$

we can define the predicate transformer $Post_A : \mathcal{P}(Q) \to \mathcal{P}(\mathcal{P}(Q))$, which operates over a smaller domain, as follows:

$$Post_A(N) = \bigcup_{a \in \Sigma} \{N_0 \mid (N, a, N_0) \in E(A)\}.$$

Then,

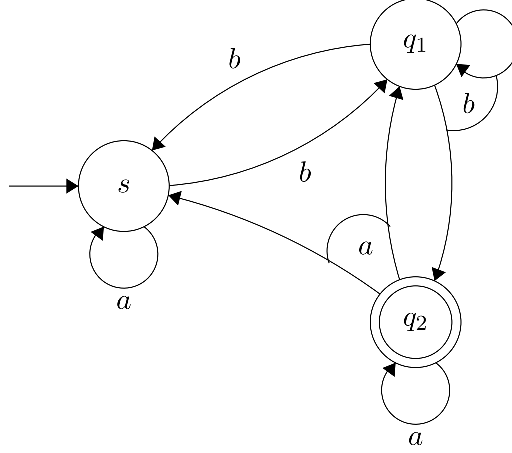

Figure 3.1: AFA A

$$post_A(X) = \bigcup_{N \in X} \{Post_A(N)\}.$$

In a similar way, it is possible to construct a analogous predicate transformer Pre_A : $\mathcal{P}(Q) \to \mathcal{P}(\mathcal{P}(Q))$, where

$$Pre_A(N) = \bigcup_{a \in \Sigma} \{N_0 \mid (N_0, a, N) \in E(A)\}.$$

Since both equations

$$\widetilde{pre}_A(X) = \overline{pre_A(\overline{X})}$$

$$\widetilde{post}_A(X) = \overline{post_A(\overline{X})}$$

hold true [4], we can compute each predicate transformer $pre_A, post_A, \widetilde{pre}_A, \widetilde{post}_A$ using simpler predicate transformers $Pre_A, Post_A$.

Example. Let A be an AFA shown in Figure 3.1. Suppose that there exists an edge (N_0, a, N_1) of A. Table 3.1 summarizes all the triplets that satisfy this property. The first column corresponds to N_0, while the first row represents N_1. The inner cells contain sets of symbols a such that $(N_0, a, N_1) \in E(A)$.

Then, we can express, for instance, $post_A(\{\{s\}, \{q_1\}\}) = \{Post_A(\{s\})\} \cup \{Post_A(\{q_1\})\} = \{\{s\}, \{q_1\}, \{s, q_1\}, \{s, q_2\}, \{q_1, q_2\}, \{s, q_1, q_2\}\}$.

	$\varnothing$	$\{s\}$	$\{q_1\}$	$\{q_2\}$	$\{s, q_1\}$	$\{s, q_2\}$	$\{q_1, q_2\}$	$\{s, q_1, q_2\}$
$\varnothing$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
$\{s\}$	$\varnothing$	$\{a\}$	$\{b\}$	$\varnothing$	$\{a, b\}$	$\{a\}$	$\{b\}$	$\{a, b\}$
$\{q_1\}$	$\varnothing$	$\{b\}$	$\varnothing$	$\varnothing$	$\{b\}$	$\{b\}$	$\{b\}$	$\{b\}$
$\{q_2\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$
$\{s, q_1\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{b\}$	$\varnothing$	$\{b\}$	$\{b\}$
$\{s, q_2\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{a\}$	$\{a\}$	$\varnothing$	$\{a\}$
$\{q_1, q_2\}$	$\varnothing$	$\varnothing$	$\{b\}$	$\varnothing$	$\{b\}$	$\varnothing$	$\{b\}$	$\{b\}$
$\{s, q_1, q_2\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

Table 3.1: Predecessors and successors in AFA A

3.2 Fixed Points of Predicate Transformers

Definition 23 *Let P be a set. Suppose that $\sqsubseteq$ is a reflexive, antisymmetric and transitive binary relation over P (partial order over P). Then, the tuple $(P, \sqsubseteq)$ is called a **partially ordered set**.*

Definition 24 *Let $(P, \sqsubseteq)$ be a partially ordered set. Consider its subset $R \subseteq P$. If there exists an element $r_0 \in R$, such that $\forall r \in R : r \sqsubseteq r_0$, then r_0 is called a **$\sqsubseteq$-maximal element** of R. Dually, r_0 is a **$\sqsubseteq$-minimal element** of R if $\forall r \in R : r_0 \sqsubseteq r$.*

Definition 25 *Let $(P, \sqsubseteq)$ be a partially ordered set and $R \subseteq P$ is a non-empty set. Suppose that $S \subseteq P$ is a set such that $S = \bigcap_{r \in R}\{p \in P \mid p \sqsubseteq r\}$. If there exists a $\sqsubseteq$-maximal element s_0 of S, then s_0 is an **infimum** of R. In what follows, the infimum of a set R will be denoted as $\sqcup R$ or $\inf_{\sqsubseteq}(R)$.*

Definition 26 *Let $(P, \sqsubseteq)$ be a partially ordered set and $R \subseteq P$ is a non-empty set. Suppose that $S \subseteq P$ is a set such that $S = \bigcap_{r \in R}\{p \in P \mid r \sqsubseteq p\}$. If there exists a $\sqsubseteq$-minimal element s_0 of S, then s_0 is an **supremum** of R. In what follows, the supremum of a set R will be denoted as $\sqcap R$ or $\sup_{\sqsubseteq}(R)$.*

Definition 27 *Let $(P, \sqsubseteq)$ be a partially ordered set. Suppose that $f : P \rightarrow P$ is a function over $(P, \sqsubseteq)$. Then, a **fixed point** of the function f is each member $p \in P$ such that $f(p) = p$.*

Definition 28 *Assume that $(P, \sqsubseteq)$ is a partially ordered set and $P_f = \{p \in P \mid f(p) = p\}$ is a set of all the fixed points of f. An element $p_0 \in P_f$ is called **the least fixed point** of f, if $p_0 = \sqcup P_f$. In what follows, we denote the least fixed point of f by $\mu X \cdot f(X)$.*

Definition 29 *Let $(P, \sqsubseteq)$ be a partially ordered set. Suppose that there exists the infimum $\sqcup R$ and the supremum $\sqcap R$ for each non-empty subset $R \subseteq P$. Let $\bot \triangleq \sqcup P$ and $\top \triangleq \sqcap P$. Then, the sextuple $(P, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ is called a **complete lattice**.*

Definition 30 *Let $(P, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ be a complete lattice. A function $f : P \rightarrow P$ is called a **monotonically increasing** function if $\forall p_0, p_1 \in P : p_0 \sqsubseteq p_1 \Rightarrow f(p_0) \sqsubseteq f(p_1)$.*

Let $M = (P, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ be a complete lattice such that the carrier P is a finite set. Suppose that $f : P \to P$ is a monotonically increasing function over M. Under these conditions, there exists an unique least fixed point of f, which results from the Knaster-Tarski theorem [9]. The least fixed point is equal to $f^k(\bot)$, where $k \in \mathbb{N} \setminus \{0\}$, such that $f^k(\bot) = f^{k+1}(\bot)$ [6].

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA and suppose that $X \subseteq \mathcal{P}(Q), X \neq \varnothing$. Then, the sextuple $(\mathcal{P}(\mathcal{P}(Q)), \subseteq, \cup, \cap, \mathcal{P}(Q), \varnothing)$ forms a complete lattice due to the fact that there exists an infimum $\inf_{\subseteq}(X) = \bigcap_{x \in X} x$ and the supremum $\sup_{\subseteq}(X) = \bigcup_{x \in X} x$ for each non-empty subset of $\mathcal{P}(Q)$.

It was proven above in Equation 7.3 that $post_A(X_0 \cup X_1) = post_A(X_0) \cup post_A(X_1)$, where $X_0, X_1 \subseteq \mathcal{P}(Q)$. In case of $X_0 \subseteq X_1$, the equation $X_0 \cup X_1 = X_1$ is true and $post_A(X_0) \subseteq post_A(X_1)$ could be also considered to be true because $post_A(X_1) = post_A(X_0 \cup X_1) = post_A(X_0) \cup post_A(X_1)$.

This implies that the predicate transformer $post_A : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$ is a monotonically increasing function.

Suppose that $I_0(A) = \bigcup_{s_0 \in S_0} [\![s_0]\!]$ and consider $f : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$ to be a function $f(X) = I_0(A) \cup post_A(X)$. It was shown in the preceding paragraphs that $post_A$ is monotonically increasing. Due to the fact that $I_0(A)$ is a constant, the function f is also monotonically increasing.

Since the sextuple $(\mathcal{P}(\mathcal{P}(Q)), \subseteq, \cup, \cap, \mathcal{P}(Q), \varnothing)$ is a complete lattice and the function f is monotonically increasing, there exists a least fixed point of f.

Thus, the equation $\mu X \cdot f(X) = f^k(\varnothing)$ (where $k \in \mathbb{N} \setminus \{0\}$ and $f^k(\varnothing) = f^{k+1}(\varnothing)$) and the fact that such a fixed point always exists for every AFA are the corollaries of the statements mentioned above. Let us show that the fixed point $\mu X \cdot f(X)$ corresponds to a set of all reachable nodes of A.

Since $post_A(\varnothing) = \bigcup_{a \in \Sigma} \{s \mid (p, a, s) \in E(A) : p \in \varnothing\} = \varnothing$, it is easily comprehensible that $f^1(\varnothing) = I_0(A) \cup post_A(\varnothing) = I_0(A)$ could be understood as a set which contains all the nodes that could be reached in 0 or less steps.

Let us assume that $f^i(\varnothing)$ is a set of all nodes of A which are reachable from some of initial nodes in $i - 1$ or less steps. Then, $f^{i+1}(\varnothing) = f \circ f^i(\varnothing) = I_0(A) \cup post_A(f^i(\varnothing))$. With use of the induction hypothesis, it is clear that $post_A(f^i(\varnothing))$ is a set of all successors of nodes which are reachable in $i - 1$ or less steps. Thus, $f^{i+1}(\varnothing)$ is a set of all nodes which are reachable in i or less steps.

Since Q is a finite set, there has to exists $k \in \mathbb{N} \setminus \{0\}$, such that $f^k(\varnothing) = f^{k+1}(\varnothing)$ and therefore, $\mu X \cdot (I_0(A) \cup post_A(X))$ is a set of all nodes which are reachable in A.

Analogous arguments could be used to prove that $\mu X \cdot (\mathcal{P}(F) \cup pre_A(X))$ is a set of all nodes of A which are terminating.

3.3 Antichains

Definition 31 *Let $(P, \sqsubseteq)$ be a partially ordered set. Consider a subset $R \subseteq P$. The function $\uparrow : \mathcal{P}(P) \to \mathcal{P}(P)$ maps R to a corresponding* ***upward-closure*** *$\uparrow R$ which is defined*

as follows: $\uparrow R = \{p \in P \mid \exists r \in R : r \sqsubseteq p\}$*. Therefore, an* ***upward-closed set*** *is such a set* $R \subseteq P$ *which satisfies a condition that* $R = \uparrow R$*.*

Definition 32 *Let* $(P, \sqsubseteq)$ *be a partially ordered set and* $R \subseteq P$*. Analogously, the map* $\downarrow: \mathcal{P}(P) \to \mathcal{P}(P)$ *represents a set transformation to the* ***downward-closure*** $\downarrow R$ *with the following definition:* $\downarrow R = \{p \in P \mid \exists r \in R : p \sqsubseteq r\}$*. A set is called a* ***downward-closed set*** *if the equality* $R = \downarrow R$ *holds.*

Definition 33 *Let* $(P, \sqsubseteq)$ *be a partially ordered set and suppose that* $R \subseteq P$*. A function* $\lceil . \rceil : \mathcal{P}(P) \to \mathcal{P}(P)$ *converts* R *to a corresponding* ***set of maximal elements*** $\lceil R \rceil$*. Formally, the set* $\lceil R \rceil = \{r \in R \mid \forall s \in R : r \sqsubseteq s \Rightarrow r = s\}$*.*

Definition 34 *Let* $(P, \sqsubseteq)$ *be a partially ordered set. and* $R \subseteq P$*. A map* $\lfloor . \rfloor : \mathcal{P}(P) \to \mathcal{P}(P)$ *transforms a domain element to a* ***set of minimal elements*** $\lfloor R \rfloor$*, where the following equation holds:* $\lfloor R \rfloor = \{r \in R \mid \forall s \in R : s \sqsubseteq r \Rightarrow s = r\}$*.*

Definition 35 *Let* $(P, \sqsubseteq)$ *be a partially ordered set. An* ***antichain*** *over* P *is a set* $R \subseteq P$ *which satisfies the condition that* $\forall r, s \in R : r \sqsubseteq s \Rightarrow r = s$*, which means that each pair of non-equal elements of* R *is* $\sqsubseteq$*-incomparable.*

Let $(P, \sqsubseteq)$ be a partially ordered set and $R \subseteq P$. It is obvious from the definition of set of minimal and maximal elements that $\lceil R \rceil$ and $\lfloor R \rfloor$ are both antichains because they cannot contain $\sqsubseteq$-comparable elements. Likewise, the equations $\uparrow R = \uparrow \lfloor R \rfloor$ and $\downarrow R = \downarrow \lceil R \rceil$ are both true [4].

Example. Let $S = \{0, 1, 2\}$. The sextuple $(\mathcal{P}(S), \subseteq, \cap, \cup, S, \varnothing)$ forms a complete lattice shown in Figure 3.2 using a Hasse diagram.

Suppose that $R = \{\{0, 1\}, \{2\}\}$. The set R is an antichain since its all elements are $\subseteq$-incomparable. Note that $R = \lfloor R \rfloor$ and $R = \lceil R \rceil$, which means that R is both set of minimal elements and set of maximal elements.

Next, we can construct a set $\uparrow R = \{\{0, 1\}, \{2\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$. Such a set contains all supersets of elements in R which belong to $\mathcal{P}(S)$. For a better illustration, each member of $\uparrow R$ is drawn in bold on the left diagram in Figure 3.2.

Dually, $\downarrow R = \{\{0, 1\}, \{2\}, \{0\}, \{1\}, \varnothing\}$. The Hasse diagram on the right in Figure 3.2 emphasises the elements of $\downarrow R$.

3.3.1 Significant Properties of Antichains

In what follows, $S_\uparrow \triangleq \{\uparrow s \mid s \subseteq \mathcal{P}(S)\}$ will be a symbolic representation of set of all possible upward-closed sets over the domain $\mathcal{P}(\mathcal{P}(S))$ in respect of the relation $\subseteq$. Analogously, $S_\downarrow \triangleq \{\downarrow s \mid s \subseteq \mathcal{P}(S)\}$ plays the role of the set of all possible downward-closed sets over the domain $\mathcal{P}(\mathcal{P}(S))$.

Similarly, we consider $S_{\lceil . \rceil} \triangleq \{\lceil s \rceil \mid s \subseteq \mathcal{P}(S)\}$ and $S_{\lfloor . \rfloor} \triangleq \{\lfloor s \rfloor \mid s \subseteq \mathcal{P}(S)\}$ to be a set of all sets of maximal elements over the domain $\mathcal{P}(\mathcal{P}(S))$ and a set of all sets of minimal

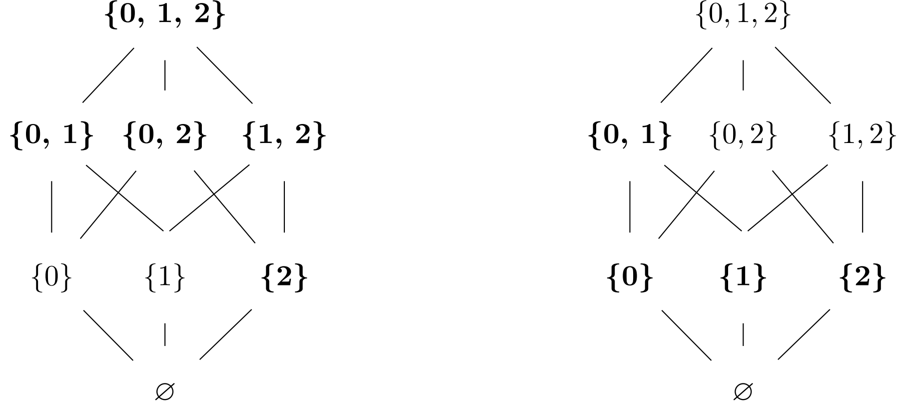

Figure 3.2: Complete lattice $(\mathcal{P}(S), \subseteq, \cap, \cup, S, \varnothing)$

elements over the domain $\mathcal{P}(\mathcal{P}(S))$, respectively. Apparently, each member of $S_{\lceil . \rceil}$ and $S_{\lfloor . \rfloor}$ is an antichain since every set of maximal and minimal elements has to be $\subseteq$-incomparable.

Consider two non-equal sets $T, U \in S_{\uparrow}$. Without loss of generality, it is possible to define $U \not\subseteq T$. Then, $\exists s \in U : s \notin T$. We know from the definition of the set of minimal elements that some set $u \subseteq s$ has to be a member of $\lfloor U \rfloor$. It is obvious that u cannot belong to $\lfloor T \rfloor$ because otherwise its superset s would be surely an element of U which is in contradiction with the preconditions. Thus, the formula $\forall U, T \in S_{\uparrow} : U \neq T \Rightarrow \lfloor U \rfloor \neq \lfloor T \rfloor$ must be true, which means that each pair of non-equal upward closed sets correspond to a pair of different sets of minimal elements.

Analogically, the formula $\forall U, T \in S_{\lfloor . \rfloor} : U \neq T \Rightarrow \uparrow U \neq \uparrow T$ holds as well. Suppose that $U \not\subseteq T$, then $\exists s \in U : s \notin T$ which signifies that either $\forall u \in T; u \not\subseteq s$ and therefore $s \notin \uparrow T$ or $\exists u \in T; u \subseteq s$ which implies that $u \notin \uparrow U$. These two branches prove the initial statement.

The essential corollary of such formulas is the fact that for a finite set S, there always exists a bijection between two domains $S_{\uparrow}$ and $S_{\lfloor . \rfloor}$. In consequence, each upward-closed set corresponds to an unique antichain which also represents exactly one upward-closed set.

Dually, we can use similar arguments to prove that for a finite set S, there also exists a bijection between two domains $S_{\downarrow}$ and $S_{\lceil . \rceil}$.

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. Assume that $X \subseteq \mathcal{P}(Q)$ and $s \in post_A(X)$. We know from the definition of $post_A$ that there exists an edge $(p, a, s) \in E(A)$, $s \in [\![\bigwedge_{q \in p} \delta(q, a)]\!]$. It is obvious from the definition of $[\![.]\!]$ that $[\![\bigwedge_{q \in p} \delta(q, a)]\!]$ has to be an upward-closed set. Since each element of $[\![\bigwedge_{q \in p} \delta(q, a)]\!]$ belongs to $post_A(X)$, the set $post_A(X)$ must be always upward-closed.

The explanation that a result of $\widetilde{pre}_A$ is always upward-closed and both pre_A and $\widetilde{post}_A$ map every input to a downward-closed would be expressed in a similar way.

Consider $X \in Q_{\lfloor . \rfloor}$, $x \in X$ and $y \in \uparrow X$ such that $x \subseteq y$. Note that X is an antichain. It was shown above that $post_A$ is a monotonically increasing function. Therefore, we can say that $post_A(X) \subseteq post_A(X \cup \{y\})$. We also know that $post_A(X \cup \{y\}) = post_A(X) \cup post_A(\{y\})$. Apparently, the inclusion $[\![\bigwedge_{q \in y} \delta(q, a)]\!] \subseteq [\![\bigwedge_{q \in x} \delta(q, a)]\!]$ can be perceived as true since $[\![\bigwedge_{q \in y} \delta(q, a)]\!] = [\![(\bigwedge_{q \in x} \delta(q, a))]\!] \cap [\![(\bigwedge_{q \in y \setminus x} \delta(q, a))]\!]$ can be shown for each $a \in \Sigma$.

In conclusion, $post_A(\{y\}) \subseteq post_A(\{x\})$. Since x is a member of X, we conclude that $post_A(X) = post_A(X \cup \{y\})$. In general, $post_A(Y) = post_A(\lfloor Y \rfloor)$ for any $Y \subseteq \mathcal{P}(Q)$.

We can also use similar arguments to show that $\widetilde{pre}_A(Y) = \widetilde{pre}_A(\lfloor Y \rfloor)$ for every $Y \subseteq \mathcal{P}(Q)$ and also $pre_A(Y) = pre_A(\lceil Y \rceil)$ and $\widetilde{post}_A(Y) = \widetilde{post}_A(\lceil Y \rceil)$.

The crucial consequence of these properties of antichains is the fact that the following equations hold for any $X \subseteq \mathcal{P}(Q)$:

$$\begin{aligned} post_A(X) &= \uparrow\lfloor post_A(\lfloor X \rfloor)\rfloor \\ \widetilde{post}_A(X) &= \downarrow\lceil \widetilde{post}_A(\lceil X \rceil)\rceil \\ pre_A(X) &= \downarrow\lceil pre_A(\lceil X \rceil)\rceil \\ \widetilde{pre}_A(X) &= \uparrow\lfloor \widetilde{pre}_A(\lfloor X \rfloor)\rfloor \end{aligned}$$

3.4 AFA Emptiness

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. It was explained above that it is possible to iteratively construct a set of all reachable and terminating nodes using the fixed point expressions and therefore decide whether the input automaton corresponds to an empty language. The previous parts of this chapter led to conclusion that there is no need to evaluate predicate transformers over each element of the set $\mathcal{P}(\mathcal{P}(Q))$ because the result of computing the functions $post_A$ and $\widetilde{pre}_A$ over $S \subseteq \mathcal{P}(Q)$ is the same as the result of computing that function over $\lfloor S \rfloor$. In the similar way, it is possible to perform a computation of pre_A and $\widetilde{post}_A$ directly on the corresponding antichain $\lceil S \rceil$ since the results are equivalent.

In addition, each element of $\mathcal{P}(Q)$ can be evaluated by $post_A$ and pre_A only once and then, the result can be reused in the future computation due to the fact that $post_A$ and pre_A can be transformed to the union of $Post_A$ and Pre_A, respectively.

Thus, the emptiness of an AFA can be decided using these formulae:

$$L(A) = \varnothing \iff \lfloor \mu X \cdot (\lfloor I_0(A) \cup post_A(X) \rfloor) \rfloor \subseteq \overline{\mathcal{P}(F)}$$

$$L(A) = \varnothing \iff \lceil \mu X \cdot (\lceil \mathcal{P}(F) \cup pre_A(X) \rceil) \rceil \subseteq \overline{I_0(A)}$$

The algorithms which ale based on these equivalences are presented below. The concrete forward language emptiness test introduced in Algorithm 4 uses the idea of computing

the reachable nodes of A, while the backward language emptiness test in Algorithm 5 summarizes the theory of terminating nodes of A.

Algorithm 4: Concrete forward language emptiness test of an AFA

```
   Input: AFA A = (Q, Σ, S_0, R, F)
   Output: True iff L(A) = ∅
1  NonFinal ← ¬P(F);
2  Previous ← ∅;
3  Next ← I_0(A);
4  while Previous ≠ Next do
5  |   Previous ← Next;
6  |   Next ← I_0(A);
7  |   for x in ⌊Previous⌋ do
8  |   |   Next ← Next ∪ {Post_A(x)};
9  |   end
10 |   if ⌊Next⌋ ⊈ NonFinal then
11 |   |   return False;
12 end
13 return ⌊Next⌋ ⊆ NonFinal;
```

Algorithm 5: Concrete backward language emptiness test of an AFA

```
   Input: AFA A = (Q, Σ, S_0, R, F)
   Output: True iff L(A) = ∅
1  Previous ← ∅;
2  Next ← Final;
3  while Previous ≠ Next do
4  |   Previous ← Next;
5  |   Next ← P(F);
6  |   for x in ⌊Previous⌋ do
7  |   |   Next ← Next ∪ {Pre_A(x)};
8  |   end
9  |   if ⌊Next⌋ ⊈ ¬S_0 then
10 |   |   return False;
11 end
12 return ⌊Next⌋ ⊆ ¬S_0;
```

(In the algorithms above, ¬P(F) and ¬S_0 denote $\overline{\mathcal{P}(F)}$ and $\overline{S_0}$.)

Chapter 4

Abstract Forward and Backward Algorithms Deciding the AFA Emptiness

The previous chapter introduced the algorithmic way to test language emptiness of an *AFA* in a concrete domain using the properties of antichains. In subsequent sections, the idea of using an *abstract domain* of an *AFA* to perform a language emptiness test will be presented.

First, the concept of a *partition* of a set, which plays the crucial role in the abstraction scheme, will be explained [1].

Next, a partition of a set of states of an *AFA* will be used to formally define an *abstract domain* of an *AFA*. We will also put emphasis on the relation between abstract and *concrete domain* of an *AFA* [4].

The possibility of a conversion between concrete and abstract domain using *concretization and abstraction functions* and remarkable algebraic properties of such functions will be discussed subsequently [4, 2].

The following paragraphs bring the concept of an *abstract AFA* constructed by an *AFA* and a partition of its set of states. They also clarify the relation between languages of these two automata.

Afterwards, both *abstract forward* and *abstract backward* algorithms will be described. The necessity of modifications of the original algorithms will be elucidated [4].

4.1 Lattice of Partitions

Definition 36 *Let S be a non-empty set. The* ***partition*** *$P \subseteq \mathcal{P}(S)$ of S is a system of sets that satisfies the following conditions: $\bigcup_{p \in P} p = S, \forall p, q \in P : p \cap q \neq \varnothing \Rightarrow p = q$ and $\forall p \in P : p \neq \varnothing$. Each element $p \in P$ will be called a* ***class*** *of the partition P.*

In other words, a partition is a system of disjunctive non-empty classes such that each member of the former set S is represented in exactly one class.

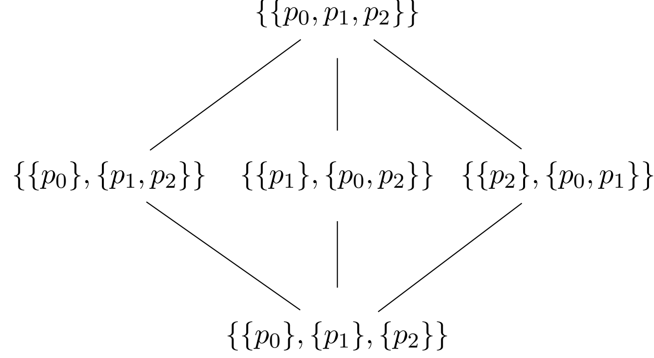

Figure 4.1: Lattice of partitions $(\mathbb{P}, \preceq, \curlywedge, \curlyvee, P_\top, P_\bot)$

Let us say that S is a non-empty set and P is its partition. Then, it is possible to define a function $P_S : S \to \mathcal{P}(S)$ which maps a single element from S to a corresponding class of P.

Assume that $\mathbb{P}$ is the set of all possible partitions of S and $P_0, P_1 \in \mathbb{P}$ are both partitions of S. We define the order relation $\preceq$ as follows: $P_0 \preceq P_1 \iff \forall p_0 \in P_0\ \exists p_1 \in P_1 : p_0 \subseteq p_1$. It is obvious from the definition that for $P_0 \preceq P_1$, the partition P_1 is *coarser* than P_0 which implies that $|P_0| \geq |P_1|$. We also say that P_0 *refines* P_1.

Let $P_\top \in \mathbb{P}$ be a $\preceq$-maximal partition of S such that $P_\top = \{S\}$. Similarly, $P_\bot \in \mathbb{P}$ is a $\preceq$-minimal partition of S due to the fact that $P_\bot = \bigcup_{s \in S} \{s\}$. Then, $\forall P \in \mathbb{P} : P_\bot \preceq P$ and $\forall P \in \mathbb{P} : P \preceq P_\top$.

Definition 37 *The* ***greatest lower bound*** *of two partitions P_0 and P_1, which is denoted by $P_0 \curlywedge P_1 \in \mathbb{P}$, is the least coarse partition of S which satisfies the properties that $P_0 \curlywedge P_1 \preceq P_0$ and $P_0 \curlywedge P_1 \preceq P_1$. Formally, $P_0 \curlywedge P_1 \triangleq \{p \mid \exists p_0 \in P_0\ \exists p_1 \in P_1 : p = p_0 \cap p_1 \wedge p \neq \varnothing\}$. Dually, the* ***least upper bound*** *of two partitions P_0 and P_1, which will be written as $P_0 \curlyvee P_1 \in \mathbb{P}$, is the coarsest partition of S such that $P_0 \preceq P_0 \curlyvee P_1$ and $P_1 \preceq P_0 \curlyvee P_1$.*

Now, we can create a sextuple $(\mathbb{P}, \preceq, \curlywedge, \curlyvee, P_\top, P_\bot)$ which forms a *complete lattice of partitions* due to the fact that for each non-empty set $T \subseteq \mathbb{P}$ there exists the infimum $\inf_\preceq(T) = \curlywedge_{p \in T}\, p$ and the supremum $\sup_\preceq(T) = \curlyvee_{p \in T}\, p$.

A possibility of creating such a complete lattice is essential for the studied algorithms. Later, it will allow us to define an *abstraction domain* of an *AFA* and we will be able to use predicate transformers directly on the classes of partitions of states of an *AFA*.

Example. Let $S = \{p_0, p_1, p_2\}$ be a set. Assume that $(\mathbb{P}, \preceq, \curlywedge, \curlyvee, P_\top, P_\bot)$ is a corresponding complete lattice of partitions of S. The complete lattice of partitions is illustrated as a Hasse diagram in Figure 4.1.

In this case, the $\preceq$-maximal partition $P_\top = \{\{p_0, p_1, p_2\}\}$, while the $\preceq$-minimal partition $P_\bot = \{\{p_0\}, \{p_1\}, \{p_2\}\}$.

Suppose that $P = \{\{p_0\}, \{p_1, p_2\}\}$ and $Q = \{\{p_1\}, \{p_0, p_2\}\}$. Then $P_S(p_0) = \{p_0\}, P_S(p_1) = \{p_1, p_2\}$. We can say that $P \curlywedge Q = P_\bot, P \curlyvee Q = P_\top$ and $P_\bot \preceq P \preceq P_\top$.

4.2 Concrete and Abstract Domain

Definition 38 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. As was stated in the previous chapter, the sextuple $M = (\mathcal{P}(\mathcal{P}(Q)), \subseteq, \cap, \cup, \mathcal{P}(Q), \varnothing)$ forms a complete lattice. In what follows, such a sextuple M will be called a **concrete domain** of the automaton A. Each element $q \in \mathcal{P}(Q)$ is then a **concrete node**.*

Definition 39 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. Assume that $\mathbb{P} \subseteq \mathcal{P}(\mathcal{P}(Q))$ is a set of all partitions of $\mathcal{P}(Q)$. The sextuple $M = (\mathcal{P}(\mathcal{P}(P)), \subseteq, \cap, \cup, \mathcal{P}(P), \varnothing)$, where $P \in \mathbb{P}$, corresponds to an **abstract domain** of the automaton A. We also use the collocation **abstract node** to each member of $\mathcal{P}(P)$.*

Let $\mathbb{B} \triangleq \{\top, \bot\}$. Due to the fact that each class of an abstract domain corresponds to a set of nodes of the concrete domain, we should be able to symbolically express such a relation. Let us define a function $C : \mathcal{P}(Q) \times \mathcal{P}(P) \rightarrow \mathbb{B}$ such that $C(q, p) \iff p = \{P_Q(q_0) \mid q_0 \in q\}$, where $p \in \mathcal{P}(P), q \in \mathcal{P}(Q)$. In this context, C is an abbreviation for *covering*.

Since the abstract domain is smaller than the concrete domain, for each $q \in \mathcal{P}(Q)$, there always exists only one $p \in \mathcal{P}(P)$ which satisfies $C(q, p) = \top$. In opposite, it is possible that for an element $p \in \mathcal{P}(P)$ there exists multiple different $q \in \mathcal{P}(Q)$, such that $C(q, p) = \top$.

Example. Let $Q = \{s_0, s_1, s_2, s_3, s_4, s_5\}, P = \{p_0 : \{s_0, s_1, s_2\}, p_1 : \{s_3, s_4\}, p_2 : \{s_5\}\}$. Under these conditions, for instance $C(\{s_3\}, \{p_1\}) = \top$ and $C(\{s_5, s_0\}, \{p_0, p_2\}) = \top$. On the other hand, $C(\{s_0\}, \{p_1\}) = \bot$ and $C(\{s_0, s_3\}, \{p_0, p_2\}) = \bot$.

4.2.1 Abstraction and Concretization Functions

Definition 40 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA, P is a partition of Q. Then, we denote by $\alpha_P : \mathcal{P}(\mathcal{P}(Q)) \rightarrow \mathcal{P}(\mathcal{P}(P))$ an **abstraction function**, which is formally defined as follows : $\alpha_P(X) = \{x_\alpha \mid \exists x_\gamma \in X : C(x_\alpha, x_\gamma)\}$.*

Definition 41 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. Assume that P is a partition of Q. We denote by $\gamma_P : \mathcal{P}(\mathcal{P}(P)) \rightarrow \mathcal{P}(\mathcal{P}(Q))$ a **concretization function**, which has the following definition: $\gamma_P(X) = \{c_\gamma \mid \exists c_\alpha \in X; C(c_\alpha, c_\gamma)\}$.*

Such a abstraction function converts all concrete nodes on its input to corresponding sets of classes of P. Likewise, the concretization function maps members of an abstract domain to appropriate concrete nodes.

Definition 42 *Let $\mathbb{X} = (X, \sqsubseteq_X), \mathbb{Y} = (Y, \sqsubseteq_Y)$ be two partially ordered sets. Suppose that both $\chi : X \rightarrow Y$ and $\psi : Y \rightarrow X$ are maps such that $\forall x \in X\ \forall y \in Y : x \sqsubseteq_X \psi(y) \iff \chi(x) \sqsubseteq_Y y$. In this case, the quadruple $(\mathbb{X}, \chi, \psi, \mathbb{Y})$ forms a **Gallois connection** [2] which will be symbolised as $\mathbb{X} \overset{(\chi,\psi)}{\longleftrightarrow} \mathbb{Y}$.*

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA and P is a partition of Q. Suppose that $X_Q \subseteq \mathcal{P}(Q)$ and $X_P \subseteq \mathcal{P}(P)$. It is obvious that both tuples $\mathbb{X}_Q = (X_Q, \subseteq)$ and $\mathbb{X}_P = (X_P, \subseteq)$ are partially

ordered sets. Firstly, assume that $X_Q \subseteq \gamma_P(X_P)$. This implies that for each concrete node $x_Q \in X_Q$, the set X_P contains an abstract node x_P such that $P_Q(x_Q) = x_P$. Then, it is easy to see that $\alpha_P(X_Q) \subseteq X_P$ since the equation $\alpha(\{x_Q\}) = x_P$ also holds. The other direction of this implication would be shown similarly.

Thus, the quadruple $(\mathbb{X}_Q, \alpha_P, \gamma_P, \mathbb{X}_P)$ forms a Gallois connection $\mathbb{X}_Q \overset{(\alpha_P,\gamma_P)}{\longleftrightarrow} \mathbb{X}_P$.

The definitions of α_P and γ_P and the fact that $\mathbb{X}_Q \overset{(\alpha_P,\gamma_P)}{\longleftrightarrow} \mathbb{X}_P$ is a Gallois connection imply few significant properties of these functions. In case of $X_0, X_1 \subseteq \mathcal{P}(Q)$ and $Y_0, Y_1 \subseteq \mathcal{P}(P)$, we can write $\alpha_P(X_0 \cup X_1) = \alpha_P(X_0) \cup \alpha_P(X_1)$ and $\gamma_P(Y_0 \cap Y_1) = \gamma_P(Y_0) \cap \gamma_P(Y_1)$ [4].

The equations $\gamma_P(Y_0 \cup Y_1) = \gamma_P(Y_0) \cup \gamma_P(Y_1)$ and $\alpha_P(X_0 \cap X_1) = \alpha_P(X_0) \cap \alpha_P(X_1)$ are also valid if X_0, X_1, Y_0 and Y_1 are all upward-closed or downward-closed [4].

Example. Let $Q = \{s_0, s_1, s_2, s_3, s_4\}$ and $P = \{p_0 : \{s_0, s_1\}, p_1 : \{s_2, s_3, s_4\}\}$ is a partition of Q. Then, $\alpha_P(\{\{s_0\}, \{s_0, s_1\}\}) = \{\{p_0\}\}$ and $\alpha_P(\{\{s_2, s_3\}\}) = \{\{p_1\}\}$. In opposite, $\gamma_P(\{\{p_1\}\}) = \{\{s_2\}, \{s_3\}, \{s_4\}, \{s_2, s_3\}, \{s_2, s_4\}, \{s_3, s_4\}, \{s_2, s_3, s_4\}\}$. This example also shows that α_P and γ_P are not inverse to each other.

4.2.2 Abstraction and Concretization Functions over Formulae

Definition 43 *Let Q be a finite set of states and P is a partition of Q. The* ***abstraction function over positive Boolean formulae*** *will by symbolised as $\alpha_P^B : B^+(Q) \to B^+(P)$. Given the input $q \in Q$, the function $\alpha_P^B(q) \triangleq P_Q(q)$ simply converts an element of concrete domain to a corresponding class of P. Suppose that $\varphi_0, \varphi_1 \in B^+(Q)$. Then, $\alpha_P^B(\varphi_0 \vee \varphi_1) = \alpha_P^B(\varphi_0) \vee \alpha_P^B(\varphi_1)$ and also $\alpha_P^B(\varphi_0 \wedge \varphi_1) = \alpha_P^B(\varphi_0) \wedge \alpha_P^B(\varphi_1)$.*

Definition 44 *Let Q be a finite set of states and P is a partition of Q. The* ***concretization function over positive Boolean formulae*** *will be shortly denoted as $\gamma_P^B : B^+(Q) \to B^+(P)$. Let us define $\gamma_P^B(p) \triangleq \bigvee_{p_0 \in p} p_0$, where $p \in P$, transforms an abstract class to the clause composed of the members of the given class. Given two formulae $\psi_0, \psi_1 \in B^+(P)$, we define $\gamma_P^B(\psi_0 \vee \psi_1) = \gamma_P^B(\psi_0) \vee \gamma_P^B(\psi_1)$ and $\gamma_P^B(\psi_0 \wedge \psi_1) = \gamma_P^B(\psi_0) \wedge \gamma_P^B(\psi_1)$.*

Let us recall that sets of valuations of each formula $\varphi \in B^+(Q)$ and $\psi \in B^+(P)$ could be represented as $[\![\varphi]\!]$ and $[\![\psi]\!]$, respectively, where $[\![\varphi]\!] \in Q_\uparrow$ and $[\![\psi]\!] \in P_\uparrow$.

Let us show that $[\![\alpha_P^B(\varphi)]\!] = \alpha_P([\![\varphi]\!])$, where $\varphi \in B^+(Q)$. Since such a positive Boolean formula could be expressed in the disjunctive normal form using only positive logical variables $q \in Q$, we can write (without loss of generality) that $\varphi = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} q_{i,j}$.

Using the definition of α_P^B, α_P and the relation between logical and set connectives emphasised within the definition of $[\![.]\!]$, we say that $[\![\alpha_P^B(\bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} q_{i,j})]\!] = [\![\bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} P(q_{i,j})]\!] = \bigcup_{i=0}^{n} \bigcap_{j=0}^{m} [\![P(q_{i,j})]\!] = \bigcup_{i=0}^{n} \bigcap_{j=0}^{m} \alpha_P([\![q_{i,j}]\!]) = \alpha_P(\bigcup_{i=0}^{n} \bigcap_{j=0}^{m} [\![q_{i,j}]\!]) = \alpha_P([\![\bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} q_{i,j}]\!]) = \alpha_P([\![\varphi]\!])$. The initial equation is proven.

Such a crucial statement allows us to define the abstract automaton in the next section.

Example. Let $Q = \{s_0, s_1, s_2, s_3, s_4, s_5\}$. $P = \{p_0 : \{s_0, s_1\}, p_1 : \{s_2, s_3\}, p_2 : \{s_4, s_5\}\}$ is a partition of Q. Assume that $\varphi = s_0 \vee (s_2 \wedge s_5)$ and $\psi = p_0 \wedge (p_1 \vee p_2)$. It is obvious

that $\varphi \in B^+(Q)$ and $\psi \in B^+(P)$. Under these conditions, it is possible to say that $\alpha_P^B(\varphi) = P(s_0) \vee (P(s_1) \wedge P(s_5)) = p_0 \vee (p_0 \wedge p_2) = p_0$. We can also demonstrate γ_P^B as follows: $\gamma_P^B(\psi) = \bigvee_{p \in p_0} p \wedge (\bigvee_{p \in p_1} p \vee \bigvee_{p \in p_2} p) = (s_0 \vee s_1) \wedge (s_2 \vee s_3 \vee s_4 \vee s_5)$. To illustrate the validity of the equation $[\![\alpha_P^B(\varphi)]\!] = \alpha_P([\![\varphi]\!])$, we can say that $[\![\alpha_P^B(\varphi)]\!] = [\![p_0]\!] = \uparrow\{\{p_0\}\}$ and also $\alpha_P([\![\varphi]\!]) = \alpha_P(\uparrow\{\{s_0\},\{s_0,s_2\}\}) = \uparrow\{\{p_0\}\}$.

4.3 Abstract Alternating Automaton

Definition 45 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA and P is a partition of Q. An* ***abstract alternating finite automaton*** *A_P^α (shortly denoted as an* ***abstract* AFA***) constructed of the concrete* AFA *A using the partition P is a quintuple $A_P^\alpha = (Q^\alpha, \Sigma, S_0^\alpha, \delta^\alpha, F^\alpha)$, where*

- *$Q^\alpha \triangleq P$ is a set of states of an abstract* AFA
- *Σ is an alphabet*
- *$S_0^\alpha \triangleq \bigcup_{s_0 \in S_0}\{P(s_0)\}$ is a set of initial states of an abstract* AFA
- *$\delta^\alpha : P \times \Sigma \to B^+(P)$ stands for an abstract transition function, which is defined as $\delta^\alpha(p, a) \triangleq \alpha_P^B(\bigvee_{p_0 \in p} \delta(p_0, a))$, where $p \in P$ and $a \in \Sigma$*
- *$F^\alpha \triangleq \bigcup_{f \in F}\{P(f)\}$ is a finite set of final states of an abstract* AFA

It is obvious from the definition that it is possible to construct various abstract automata from the concrete automaton depending on the given partition of states. This allows us to separate the original automaton to several parts, temporarily ignore the behaviour within each part and inspect the relations between various parts of the automaton. The example below demonstrates the construction of several different abstract automata from the given concrete automaton in detail.

Since an abstract *AFA* is in fact still an *AFA*, each additional definition presented in Section 2.3 is reusable in context of an abstract *AFA*.

Example. Let $A = (Q, \Sigma, S_0, \delta, F)$ be a concrete *AFA*, such that $Q = \{s, q_1, q_2, q_3, f\}$, $\Sigma = \{a, b\}$, $s_0 = \{s\}$ and $F = \{f\}$. The transition relation is defined as follows:

$$\begin{aligned}
&\delta(s, a) = q_1 \wedge q_2 \\
&\delta(q_1, b) = q_2, \delta(q_1, a) = q_3 \\
&\delta(q_2, a) = q_3 \\
&\delta(q_3, a) = q_3 \wedge f, \delta(q_3, b) = f.
\end{aligned}$$

The diagram of the concrete AFA A is shown in Figure 4.2 on left. Next, we have three partitions of Q, which are chosen as follows:

$$\begin{aligned}
&P_0 = \{p_0 : \{s, q_1, q_2\}, p_1 : \{q_3, f\}\} \\
&P_1 = \{p_0 : \{s, q_1, q_2, q_3\}, p_1 : \{f\}\} \\
&P_2 = \{p_0 : \{s, q_1\}, p_1 : \{q_2, q_3\}, p_2 : \{f\}\}
\end{aligned}$$

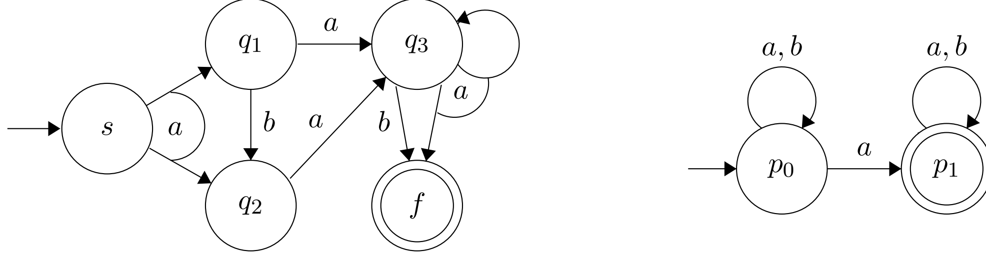

Figure 4.2: Concrete AFA A and abstract AFA $A^{\alpha}_{P_0}$

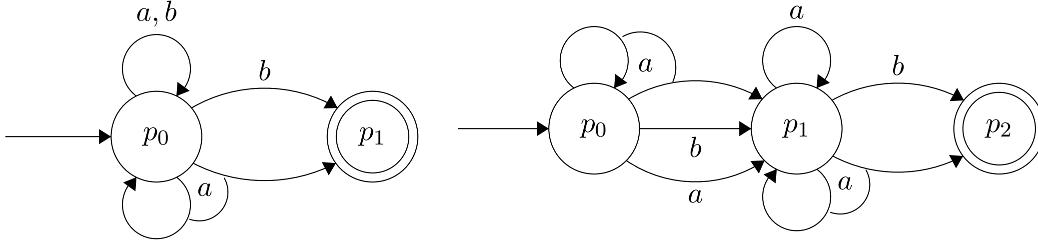

Figure 4.3: Abstract AFA $A^{\alpha}_{P_1}$ and abstract AFA $A^{\alpha}_{P_2}$

Then, we can define three various abstract AFAs $A^{\alpha}_{P_0}$ (shown in Figure 4.2 on right), $A^{\alpha}_{P_1}$ (shown in Figure 4.3 on left) and $A^{\alpha}_{P_2}$ (shown in Figure 4.3 on right).

Let us demonstrate a construction of $A^{\alpha}_{P_0}$ in detail. An abstract automaton $A^{\alpha}_{P_0}$ is a quintuple $A^{\alpha}_{P_0} = (Q^{\alpha}, \Sigma, S^{\alpha}_0, \delta^{\alpha}, F^{\alpha})$, where $Q^{\alpha} = P_0 = \{p_0 : \{s, q_1, q_2\}, p_1 : \{q_3, f\}\}$, which implies that $A^{\alpha}_{P_0}$ is composed of two states. Next, $S^{\alpha}_0 = \bigcup_{s_0 \in S_0}\{P(s_0)\} = \{p_0\}$ and $F^{\alpha} = \bigcup_{f \in F}\{P(f)\} = \{p_1\}$. The function values of δ^{α} would be computed for each tuple $(p, x) \in \{\{p_0, p_1\} \times \{a, b\}\}$. The following equation shows the computation of $\delta^{\alpha}(p_0, a)$ in detail.

$$
\begin{aligned}
&\delta^{\alpha}(p_0, a) \\
&= \alpha^{B}_{P}(\delta(s, a) \vee \delta(q_1, a) \vee \delta(q_2, a)) \\
&= \alpha^{B}_{P}((q_1 \wedge q_2) \vee q_3 \vee q_3) \\
&= (P(q_1) \wedge P(q_2)) \vee P(q_3) \\
&= (p_0 \wedge p_0) \vee p_1 \\
&= p_0 \vee p_1
\end{aligned}
$$

Similarly, such a procedure leads to the results $\delta^{\alpha}(p_0, b) = p_0, \delta^{\alpha}(p_1, a) = p_1, \delta^{\alpha}(p_1, b) = p_1$.

4.3.1 Language of an Abstract AFA

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA* and P is a partition of Q. Using the definition of abstract *AFA* from the previous section, we are now able to construct an automaton A^{α}_{P}, which represents an abstraction of A. Since the abstract *AFA* A^{α}_{P} is composed of less or

equal amount of states than A, it could possibly make the whole process of the language emptiness test easier by direct inspecion of the abstract automaton. Thus, it is necessary to find out the connection between language of the concrete *AFA* $L(A)$ and language of the abstract *AFA* $L(A^\alpha_P)$. The following paragraphs prove that $L(A) \subseteq L(A^\alpha_P)$. Since the proof presented in [4] is based on a graph-like representation of an *AFA*, we figured out a proof which corresponds to our definition of a run of an *AFA*.

Suppose that $w = a_0a_1\ldots a_n$, where $\forall i \in \mathbb{N} : i \leq n \Rightarrow a_i \in \Sigma$, is a string, which is accepted by A. We say that $w \in L(A)$. Then, there exists a run $\tau_0 \vdash^*_{E(A)} \tau_n$ of A, such that $\tau_0 = (N_0, w)$ and $\tau_n = (N_f, \epsilon)$, where N_0 is an initial node of A and N_f is a final node of A.

We also conclude from the definition of a transition that each transition $\tau_i \vdash \tau_{i+1}$, where $i \in \mathbb{N} \wedge i < n$, is equivalent to $(N_i, a_iw_0) \vdash (N_{i+1}, w_0)$, such that $(N_i, a_i, N_{i+1}) \in E(A)$ and w_0 is a suffix of w.

Now, we try to decide whether there exists such a run $\tau^\alpha_0 \vdash^*_{E(A)} \tau^\alpha_n$ of A^α_P, where $\tau^\alpha_0 = (N^\alpha_0, w)$ and $\tau^\alpha_n = (N^\alpha_f, \epsilon)$ where N^α_0 is an initial node of A^α_P and N^α_f is a final node of A^α_P. If so, then $w \in L(A^\alpha_P)$.

Suppose that $N^\alpha_i, N^\alpha_{i+1} \subseteq \mathcal{P}(P)$ are both abstract nodes created of concrete nodes N_i and N_{i+1}, respectively, which means that $\alpha_P(\{N_i\}) = \{N^\alpha_i\}$ and $\alpha_P(\{N_{i+1}\}) = \{N^\alpha_{i+1}\}$. It follows from the definition of an edge of A^α_P that $(N^\alpha_i, a_i, N^\alpha_{i+1}) \in E(A^\alpha_P)$ as soon as $N^\alpha_{i+1} \in [\![\bigwedge_{p \in N^\alpha_i} \delta^\alpha(p, a_i)]\!]$. We can derive $[\![\bigwedge_{p \in N^\alpha_i} \delta^\alpha(p, a_i)]\!] = [\![\bigwedge_{p \in N^\alpha_i} \alpha^B_P(\bigvee_{q \in p} \delta(q, a_i))]\!] = [\![\alpha^B_P(\bigwedge_{p \in Q^\alpha_i} \bigvee_{q \in p} \delta(q, a_i))]\!] = \alpha_P([\![\bigwedge_{p \in Q^\alpha_i} \bigvee_{q \in p} \delta(q, a_i)]\!])$.

The definition of $E(A)$ implies that $N_{i+1} \in [\![\bigwedge_{q \in N_i} \delta(q, a_i)]\!]$, or equivalently $\{N_{i+1}\} \subseteq [\![\bigwedge_{q \in Q_i} \delta(q, a_i)]\!]$. Since $[\![\bigwedge_{q \in N_i} \delta(q, a_i)]\!] \subseteq [\![\bigwedge_{p \in N^\alpha_i} \bigvee_{q \in p} \delta(q, a_i)]\!]$, we conclude from transitivity that $\{N_{i+1}\} \subseteq [\![\bigwedge_{p \in N^\alpha_i} \bigvee_{q \in p} \delta(q, a_i)]\!]$. The function α_P is monotonically increasing in relation to $\subseteq$. Thus, $\{N^\alpha_{i+1}\} \subseteq \alpha_P([\![\bigwedge_{p \in N^\alpha_i} \bigvee_{q \in p} \delta(q, a_i)]\!])$, which signifies that $(N^\alpha_i, a_i, N^\alpha_{i+1}) \in E(A^\alpha_P)$ and $(N^\alpha_i, a_iw_0) \vdash (N^\alpha_{i+1}, w_0)$ is a valid transition of A^α_P which can be denoted as $\tau^\alpha_i \vdash \tau^\alpha_{i+1}$.

Due to the fact that $w \in L(A^\alpha_P)$, we deduce that $L(A) \subseteq L(A^\alpha_P)$. This essential finding signifies that an abstract *AFA* over-approximates an original concrete *AFA*. Note that the emptiness of an abstract *AFA* indicates that its corresponding concrete *AFA* is also empty, owing to fact that $\varnothing$ is an only subset of itself.

Example. Suppose that A, P_0, P_1 and P_2 have the same meaning as in Example 4.3.1. Inspecting the automaton A, we would find out that $w = aab$ is the only string accepted by A. Therefore, $L(A) = \{aab\}$.

In context of $A^\alpha_{P_0}$, we are able to discover the run $(\{p_0\}, aab) \vdash (\{p_0\}, ab) \vdash (\{p_1\}, b) \vdash (\{p_1\}, \epsilon)$ which shows that $aab \in L(A^\alpha_P)$ and therefore $L(A) \subseteq L(A^\alpha_{P_0})$. Analogously, we are capable of constructing such runs $(\{p_0\}, aab) \vdash (\{p_0\}, ab) \vdash (\{p_0\}, b) \vdash (\{p_1\}, \epsilon)$ and $(\{p_0\}, aab) \vdash (\{p_1\}, ab) \vdash (\{p_1\}, b) \vdash (\{p_2\}, \epsilon)$ which show that $L(A) \subseteq L(A^\alpha_{P_1})$ and $L(A) \subseteq L(A^\alpha_{P_2})$, respectively.

4.3.2 Predicate Transformers over Abstract AFA

We are already able to decide whether an automaton A corresponds to an empty language using predicate transformers. In the context of an abstract domain, it seems to be necessary to perform γ_P conversions back to the concrete domain whenever the predicate transformers are required. Since the α_P- and γ_P-conversions are computationally expensive, its crucial to find more effective way to avoid explicit $\alpha_P \circ post_A \circ \gamma_P$ and $\alpha_P \circ pre_A \circ \gamma_P$ operations.

The following properties of predicate transformers are the crucial result of [4]:

$$post_{A^\alpha} = \alpha_P \circ post_A \circ \gamma_P$$
$$pre_{A^\alpha} = \alpha_P \circ pre_A \circ \gamma_P.$$

Such a discovery allows us to perform predicate transformers directly over the abstract automata and inspect their behaviour without a necessity of provide α_P- and γ_P-conversions within each step of exploring them.

4.4 Representability of Concrete Nodes in an Abstract Domain

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA* and P is a partition of Q. It was emphasised earlier that α_P and γ_P are not inverse to each other. Therefore, given a partition P, a lot of concrete nodes of A cannot be represented in the abstract domain without loss of precision because it would not be possible to convert them correctly back to the concrete domain in general.

Suppose that $X \subseteq \mathcal{P}(Q)$ is a set of concrete nodes of A. Let us say that nodes of X are the only concrete nodes we want to work with in the abstract domain. Then, when creating an abstract *AFA*, we can choose such a partition of Q which guarantees us that the set of concrete nodes X would be convertible between a concrete and abstract domain without loss of precision. At the same time, we do not care about the behaviour of concrete nodes which are not members of X.

This section summarizes the theory of computing the *coarsest* ($\curlyvee$-maximal) partition presented in [4] which complies with the requirements explained above.

Definition 46 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an* AFA*. Assume that $X \subseteq \mathcal{P}(Q)$. In what follows, the composite function $\gamma_P^+ : \mathcal{P}(\mathcal{P}(Q)) \to \mathcal{P}(\mathcal{P}(Q))$, where $\gamma_P^+ \triangleq \gamma_P \circ \alpha_P$, will be referred as an* ***concretization closure****.*

Definition 47 *Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA and P is a partition of Q. Consider $X \subseteq \mathcal{P}(Q)$. We say that X is* ***representable*** *in the abstract domain of A_P^α if X is a fixed point of the concretization closure, which means that $\gamma_P^+(X) = X$. Under these conditions, $\gamma_P \circ \alpha_P(X) = X$.*

Definition 48 *Let S be a finite set, $X \subseteq \mathcal{P}(S)$ is an upward-closed set and $s \in S$. The* X***-neighbours*** *of s are all members of the set $\mathcal{N}_X(s) = \{y \setminus \{s\} \mid y \in \lfloor X \rfloor \wedge s \in y\}$.*

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA*, P is a partition of Q and $X \subseteq \mathcal{P}(Q)$ is an upward-closed set of concrete nodes. X is representable in the abstract domain $\mathcal{P}(\mathcal{P}(P))$ if and only if $\forall q, s \in Q : P_Q(s) = P_Q(q) \Rightarrow \mathcal{N}_X(s) = \mathcal{N}_X(q)$. If we create such a partition that each class consists of elements of Q which are X-neighbours, then we obtain a $\curlyvee$-maximal partition $P_\curlyvee$ of Q such that X is representable in the abstract domain $\mathcal{P}(\mathcal{P}(P_\curlyvee))$. Thus, the automaton $A^{\alpha}_{P_\curlyvee}$ will be the smallest which is able to represent X without loss of precision.

Given a downward-closed set $Y \subseteq \mathcal{P}(Q)$, it is possible to utilize the crucial property that $\gamma^+_P(Y) = Y \iff \gamma^+_P(\overline{Y}) = \overline{Y}$ and simply reuse the arguments shown above for a complement of Y, which is upward-closed.

4.5 The Abstract Algorithms

In comparison to the concrete forward and backward algorithms for testing *AFA* emptiness, the abstract algorithms presented in [4] use the idea of an abstract domain. Similarly to the concrete forward and backward algorithms, the abstract algorithms decide whether there exists a reachable final state or whether one of the initial states is terminating. In case of the abstract forward algorithms, the set Z_i, which is iteratively built during the whole process, represents in i^{th} iteration an over-approximation of all concrete nodes which cannot reach any final node within i or less steps. By contrast, the abstract backward algorithm looks for all the states which are not reachable from none of initial states within i or less steps and stores its over-approximation in Z_i as well.

Due to the fact that it is necessary to guarantee that a set Z_i is always representable in an abstract domain, both algorithms use the $\curlyvee$-maximal partition $P_\curlyvee$ of Q to create an abstract *AFA* $A^{\alpha}_{P_\curlyvee}$, such that Z_i is representable in $\mathcal{P}(\mathcal{P}(P_\curlyvee))$.

This procedure allows us to perform the computation of fixed points of predicate transformers directly over the abstract domain. This process is obviously less computationally complex since the abstract automaton consists of less states than the proper concrete automaton. Nevertheless, the set Z_i is an over-approximation, so it can be inevitable to perform its computation multiple times over various abstract *AFA*s to get a sufficient result.

This section introduces both abstract algorithms originally presented in [4] and clarifies all changes which had to be done by the author of this thesis to make them work properly.

4.5.1 The Abstract Forward Algorithm

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA*. As was stated above, the abstract forward algorithm computes a set Z_i, which corresponds to an over-approximation of all concrete nodes which cannot reach any final node within i or less steps. Since final nodes are exactly those nodes which are reachable from themselves in 0 steps, the very first abstract *AFA* is created using the partition $P_0 = \{F, Q \setminus F\}$ with exactly two clases *[step* 5*]* and the corresponding Z_0 representable in $\mathcal{P}(\mathcal{P}(P_0))$ is chosen as $Z_0 = \mathcal{P}(Q) \setminus \mathcal{P}(F)$ *[step* 6*]*. Due to the fact that according to the definition of a partition any class of a partition cannot be empty, it is neccessary to chceck whether $F = \varnothing$ or $Q \setminus F = \varnothing$ *[step* $1 - 4$*]*. In these cases, the *AFA*

emptiness could be trivially decided without any additional computation. These lines were added to the original algorithm presented in [4] to solve these extreme cases.

Next, the main loop will be executed. First, it will be checked whether there exists an initial node of A which is not part of Z_i *[step 8]*. If so, the automaton is surely not empty, because this node is able to reach a final node in i or less steps.

Otherwise, the proper abstract automaton $A^{\alpha}_{P_i}$ will be constructed *[step 10]* and subsequently, a set R_i will be computed. This set should contain all the abstract nodes which are reachable from some of the initial nodes and which correspond to a concrete node in Z_i. In other words, if there exists a abstract node $r \in R_i$, then r is reachable in $A^{\alpha}_{P_i}$ and all the concrete nodes $\gamma_P(\{r\})$ cannot reach none of the final states in A within i or less steps *[step 11]*.

If there does not exist any successor of any node of R_i which cannot leave the set of abstract nodes corresponding to Z_i in one step, then the automaton is surely empty. This means that both abstract nodes of R_i and their successors cannot reach any of the final nodes in i or less steps in the concrete domain *[step 12]*.

In the original algorithm presented in [4], this step preceded the 8th step which determines whether the automaton is not empty. To ensure generality of presented algorithms, the order of these lines was changed by the author of this thesis. In some cases, the condition on line 12 is satisfied although the automaton is not empty, which leads to wrong results.

Next, the set Z_{i+1} will be computed as all the nodes of R_i converted to a concrete domain which are also controlled predecessors of $\gamma_{P_i}(R_i)$ in respect of A.

Finally, the new abstract domain will be defined as the coarsest partition of Q which is able to represent Z_{i+1}.

Algorithm 6: Abstract forward algorithm deciding the *AFA* emptiness

```
   Input: AFA A = (Q, Σ, S_0, δ, F)
   Output: True iff L(A) = ∅
 1 if F = Q then
 2 |  return False;
 3 if F = ∅ then
 4 |  return True;
 5 P_0 ← {F, Q \ F};
 6 Z_0 ← P(Q) \ P(F);
 7 for i in ℕ do
 8 |  if I_0(A) ⊈ Z_i then
 9 |  |  return False;
10 |  A^α_{P_i} ← (Q^α, Σ, S^α_0, δ^α, F^α);
11 |  R_i ← μx · (I_0(A) ∪ post_{A^α_{P_i}}(x)) ∩ α_{P_i}(Z_i);
12 |  if post_{A^α_{P_i}}(R_i) ⊆ α_{P_i}(Z_i) then
13 |  |  return True;
14 |  Z_{i+1} ← γ_{P_i}(R_i) ∩ p̃re_A(γ_{P_i}(R_i));
15 |  P_{i+1} ← ⋎{P ∈ ℙ | γ^+_P(Z_{i+1}) = Z_{i+1}};
16 end
```

4.5.2 The Abstract Backward Algorithm

Due to the fact that the abstract backward algorithm is dual to the abstract forward one presented above, it will be described more briefly.

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA*. The abstract backward algorithm computes a set Z_i, which corresponds to an over-approximation of all concrete nodes which are not reachable from any initial state within i or less steps. Then, the initial partition is equal to $P_0 = \{S_0, Q \setminus S_0\}$ with two classes since the initial nodes are the only nodes which can be reached from initial nodes in 0 steps *[step* 5*]*. Analogously, we define $Z_0 = \mathcal{P}(Q) \setminus I_0(A)$ *[step* 6*]*.

Next, the main loop will be executed. Before anything else, it will be checked if there exists a concrete final node that is reachable from one of the initial nodes in i or less steps. If so, the automaton A is not empty *[step* 8*]*.

Then, the corresponding abstract *AFA* will be created using the partition P_i *[step* 10*]* and the set of terminating nodes which are part of $\alpha_{P_i}(Z_i)$ will be computed as R_i *[step* 11*]*.

If none of the abstract nodes in R_i has a predecessor whose corresponding abstract node is reachable from one of the initial nodes in i or less steps, the given *AFA* is surely empty *[step* 12*]*.

Finally, Z_i and the new coarsest partition which can represent Z_i will be computed *[step* $14 - 15$*]*.

Algorithm 7: Abstract backward algorithm deciding the *AFA* emptiness

---

Input: AFA $A = (Q, \Sigma, S_0, \delta, F)$
Output: True iff $L(A) = \varnothing$

```
1  if F = Q then
2  |  return False;
3  if F = ∅ then
4  |  return True;
5  P_0 ← {S_0, Q \ S_0};
6  Z_0 ← 𝒫(Q) \ I_0(A);
7  for i in ℕ do
8  |  if 𝒫(F) ⊈ Z_i then
9  |  |  return False;
10 |  A^α_{P_i} ← (Q^α, Σ, s^α_0, δ^α, F^α);
11 |  R_i ← μx · (𝒫(F^α) ∪ pre_{A^α_{P_i}}(x)) ∩ α_{P_i}(Z_i);
12 |  if pre_{A^α_{P_i}}(R_i) ⊆ α_{P_i}(Z_i) then
13 |  |  return True;
14 |  Z_{i+1} ← γ_{P_i}(R_i) ∩ ~post_A(γ_{P_i}(R_i));
15 |  P_{i+1} ← ⋎{P ∈ ℙ | γ^+_P(Z_{i+1}) = Z_{i+1}};
16 end
```

Chapter 5

Abstract Bidirectional Algorithms Deciding the AFA Emptiness

In Chapter 4, the abstract forward and backward algorithms, which decide the *AFA* emptiness, were presented. Let us recall that both mentioned algorithms iteratively build a set Z_i, which contains all concrete nodes which cannot reach a final node in i or less steps, or all concrete nodes which are not reachable from one of the initial nodes in i or less steps, respectively. In each iteration of those algorithms, such a set is represented in an abstract domain to reduce the number of states of an *AFA*. To achieve this goal, the Υ-maximal partition of a state set which is able to represent Z_i in the abstract domain is used.

In what follows, the author of this thesis present his own idea of the possibility of computing a set Z_i in forward and backward fashion at the same time. This chapter also introduces newly proposed *abstract bidirectional* algorithms, which were made up by the author of this thesis, and discusses the motivation of using a bidirectional approach.

5.1 Motivation of Using a Bidirectional Approach

Suppose that A_1 and A_2 are both *AFA*s shown in Figure 5.1 and Figure 5.2, respectively. Notice that both mentioned *AFA*s are empty. We can use both Algorithm 6 and Algorithm 7 to verify this statement.

The Table 5.1 summarizes some of the important sets which are computed during the process of using an abstract forward algorithm to decide emptiness of A_1, while Table 5.2 contains corresponding sets which are created by abstract backward algorithm to decide emptiness of the same *AFA*. Let us recall that an *AFA* is considered to be empty as soon as $\alpha_{P_i}(Z_i) \subseteq post_{A^{\alpha}_{P_i}}(R_i)$ or $\alpha_{P_i}(Z_i) \subseteq pre_{A^{\alpha}_{P_i}}(R_i)$, respectively.

Note that the abstract forward algorithm required only one iteration of its main loop to decide emptiness of A_1. In opposite, the abstract backward algorithm performed 4 iterations of its main loop. Since it is necessary to create an abstract automaton, compute a fixed point of a predicate transformer over the abstract *AFA*, recompute Z_i and create a new partition during each iteration of these algorithms, it is possible to say that Algorithm 6 decided emptiness of A_1 more effectively than Algorithm 7.

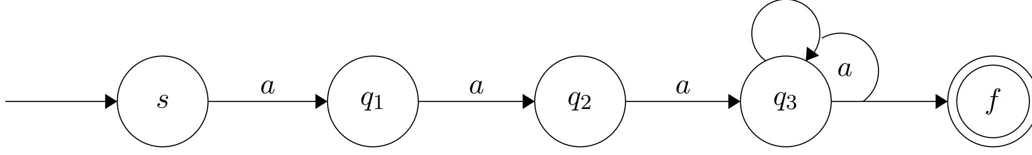

Figure 5.1: AFA A_1

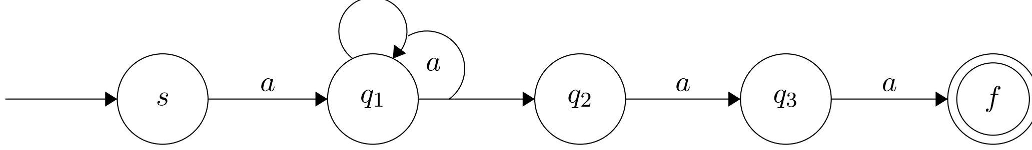

Figure 5.2: AFA A_2

By contrast, the Tables 5.3 and 5.4 show the process of deciding emptiness of A_2 using the same algorithms. In this case, the abstract backward algorithm required less iterations than the abstract forward algorithm to find the correct result.

i	Z_i	$\alpha_{P_i}(Z_i)$	$post_{A^{\alpha}_{P_i}}(R_i)$
0	$\uparrow\{\{s\},\{q_1\},\{q_2\},\{q_3\}\}$	$\uparrow\{\{p_1\}\}$	$\uparrow\{\{p_1\}\}$

Table 5.1: Deciding emptiness of A_1 using abstract forward algorithm

This observation signifies that there exists a class of AFAs whose emptiness is decided more effectively using Algorithm 6 than using Algorithm 7 and vice versa.

Let A_3 be another AFA, such that $L(A_3) \neq \varnothing$. In this case, both abstract forward and abstract backward algorithms iteratively reduce the size of a set Z_i, until there exists an initial or a final node which was removed from this set. If we use a bidirectional approach, we possibly could find a node which is reachable in i or less steps and which is terminating in i or less steps at the same moment. In this case, it is possible that the result will be found faster since we could reuse plenty of previously computed information while computing two different sets Z_i in both directions. This idea will be described in following sections more in detail.

Thus, the main idea of the bidirectional approach is as follows. Firstly, it could be a meaningful compromise between both former abstract algorithms in case of an empty automaton on input. Secondly, the bidirectional approach could possibly reuse some information computed before in case of non-empty automaton and work more efficiently with it.

5.2 Forward-like Abstract Bidirectional Algorithm

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. Let us recall that Algorithm 6 iteratively computes a set Z_i^f, which is an over-approximation of all nodes of A which cannot reach a final node in i or

i	Z_i	$\alpha_{P_i}(Z_i)$	$pre_{A^{\alpha}_{P_i}}(R_i)$
0	$\downarrow\{\{q_1, q_2, q_3, f\}\}$	$\downarrow\{\{p_2\}\}$	$\downarrow\{\{p_1, p_2\}\}$
1	$\downarrow\{\{q_2, q_3, f\}\}$	$\downarrow\{\{p_2\}\}$	$\downarrow\{\{p_1, p_2\}\}$
2	$\downarrow\{\{q_3, f\}\}$	$\downarrow\{\{p_2\}\}$	$\downarrow\{\{p_1, p_2\}\}$
3	$\downarrow\{\{f\}\}$	$\downarrow\{\{p_2\}\}$	$\downarrow\{\{\}\}$

Table 5.2: Deciding emptiness of A_1 using abstract backward algorithm

i	Z_i	$\alpha_{P_i}(Z_i)$	$post_{A^{\alpha}_{P_i}}(R_i)$
0	$\uparrow\{\{s\}, \{q_1\}, \{q_2\}, \{q_3\}\}$	$\uparrow\{\{p_1\}\}$	$\uparrow\{\{p_1\}, \{p_2\}\}$
1	$\uparrow\{\{s\}, \{q_1\}, \{q_2\}, \{q_3, f\}\}$	$\uparrow\{\{p_1\}, \{p_2, p_3\}\}$	$\uparrow\{\{p_1\}, \{p_2\}\}$
2	$\uparrow\{\{s\}, \{q_1\}, \{q_2, q_3\}, \{q_2, q_4\}, \{q_3, f\}\}$	$\uparrow\{\{p_1\}, \{p_2, p_3\}, \{p_2, p_4\}, \{p_3, p_4\}\}$	$\uparrow\{\{p_1\}\}$

Table 5.3: Deciding emptiness of A_2 using abstract forward algorithm

less steps. Analogously, the Algorithm 7 computes a set Z^b_i, which is an over-approximation of all nodes of A which cannot be reached from any of the initial nodes in i or less steps.

We can define $Z^{\rightarrow}_i$ to be a set of all concrete nodes of A which are reachable in i or less steps and $Z^{\leftarrow}_i \triangleq Z^f_i$. Notice that the relation $Z^{\rightarrow}_i \not\subseteq Z^{\leftarrow}_i$ holds if the given automaton is not empty. We can conclude from that relation that $\exists x \in \mathcal{P}(Q) : x \in Z^{\rightarrow}_i \wedge x \notin Z^{\leftarrow}_i$, which means that x is reachable from an initial node in i or less steps and it is possible to reach a final node of A from x in i or less steps.

It follows from the presented algorithms that $Z^{\leftarrow}_i$ is always an upward-closed set. We can conclude from the definition of $post_A$ that $Z^{\rightarrow}_i$ is also upward-closed. We are already able to compute coarsest partitions $P^{\rightarrow}_i, P^{\leftarrow}_i$ of Q such that $Z^{\rightarrow}_i$ is representable in $A^{\alpha}_{P^{\rightarrow}_i}$ and $Z^{\leftarrow}_i$ is representable in $A^{\alpha}_{P^{\leftarrow}_i}$. Due to the fact that we require to represent both $Z^{\leftarrow}_i$ and $Z^{\rightarrow}_i$ in the same abstract domain to avoid working with two different *AFA*s in each iteration, it is neccessery to find a corresponding partition $P^{\leftrightarrow}_i$ of Q, such that both $Z^{\leftarrow}_i$ and $Z^{\rightarrow}_i$ are representable in $A^{\alpha}_{P^{\leftrightarrow}_i}$.

Suppose that $\mathbb{P}$ is a set of all partitions of Q and $X, Y \subseteq \mathcal{P}(Q)$ are both upward-closed sets of concrete nodes of A. We know from before that $P_X = \curlyvee\{P \in \mathbb{P} \mid \gamma^+_P(X) = X\}$ and $P_Y = \curlyvee\{P \in \mathbb{P} \mid \gamma^+_P(Y) = Y\}$ are both the coarsest partitions which can represent X, Y, respectively, in an abstract domain of A.

Let us define $P_{X \curlywedge Y} = P_X \curlywedge P_Y$. Since $(\mathbb{P}, \preceq, \curlyvee, \curlywedge, P_\top, P_\bot)$ forms a complete lattice of partitions, it is easy to see that $P_{X \curlywedge Y} \preceq P_X$ and also $P_{X \curlywedge Y} \preceq P_Y$. The question is whether X is representable in the abstract domain $\mathcal{P}(\mathcal{P}(P_{X \curlywedge Y}))$, if we know that it is representable in the abstract domain $\mathcal{P}(\mathcal{P}(P_X))$.

Since we could derive $\gamma^+_{P_X}(\uparrow X) = \gamma_{P_X}(\alpha_{P_X}(\uparrow X)) = \gamma_{P_X}(\alpha_{P_X}(\uparrow\lfloor X \rfloor)) = \uparrow\gamma_{P_X}(\alpha_{P_X}(\lfloor X \rfloor)) = \uparrow\gamma_{P_X}(\cup_{x \in \lfloor X \rfloor}\alpha_{P_X}(\{x\})) = \uparrow\cup_{x \in \lfloor X \rfloor}\gamma_{P_X}(\alpha_{P_X}(\{x\}))$ and also $\uparrow X = \uparrow\cup_{x \in \lfloor X \rfloor}\{x\}$, we can conclude that $\uparrow\cup_{x \in \lfloor X \rfloor}\gamma_{P_X}(\alpha_{P_X}(\{x\})) = \uparrow\cup_{x \in \lfloor X \rfloor}\{x\}$ due to the fact that X is representable in $\mathcal{P}(\mathcal{P}(P_X))$.

Note that $\lfloor X \rfloor$ is an antichain, which also means that $\forall x \in \lfloor X \rfloor : \uparrow\gamma_{P_X}(\alpha_{P_X}(\{x\})) = \uparrow\{x\}$.

It is possible to say that $\forall x \in \lfloor X \rfloor : \gamma^+_{X \curlywedge Y}(\{x\}) \subseteq \gamma^+_X(\{x\})$ because $P_{X \curlywedge Y} \preceq P_X$, which also means that $\forall x \in \lfloor X \rfloor : \uparrow\gamma^+_{X \curlywedge Y}(\{x\}) \subseteq \uparrow\gamma^+_X(\{x\})$ and $\forall x \in \lfloor X \rfloor : \uparrow\gamma^+_{X \curlywedge Y}(\{x\}) \subseteq \uparrow\{x\}$.

i	Z_i	$\alpha_{P_i}(Z_i)$	$re_{A^{\alpha}_{P_i}}(R_i)$
0	$\downarrow\{\{q_1\},\{q_2\},\{q_3\},\{f\}\}$	$\downarrow\{\{p_1\}\}$	$\downarrow\{\{p_1,p_2\}\}$
1	$\downarrow\{\{q_2\},\{q_3\},\{f\}\}$	$\downarrow\{\{p_1\},\{p_2,p_3\}\}$	$\downarrow\{\{p_1\}\}$

Table 5.4: Deciding emptiness of A_2 using abstract backward algorithm

It follows from the definition of Gallois connection that $\forall x \in \lfloor X \rfloor : \uparrow\{x\} \subseteq \uparrow\gamma^{+}_{X \curlywedge Y}(\{x\})$, which means that we can conclude $\forall x \in \lfloor X \rfloor : \uparrow\{x\} = \uparrow\gamma^{+}_{X \curlywedge Y}(\{x\}) = \uparrow\gamma^{+}_{X}(\{x\})$ and in general, $\gamma^{+}_{X}(X) = \gamma^{+}_{X \curlywedge Y}(X)$.

Since we can simply substitute X for Y, we also know that $\gamma^{+}_{Y}(Y) = \gamma^{+}_{X \curlywedge Y}(Y)$, which means that both sets X and Y are representable in the abstract domain $\mathcal{P}(\mathcal{P}(P_{X \curlywedge Y}))$.

Such an observation allows us to say that both $Z^{\leftarrow}_i$ and $Z^{\rightarrow}_i$ and representable in $A^{\alpha}_{P^{\leftrightarrow}_i}$, where $P^{\leftrightarrow}_i \triangleq P^{\leftarrow}_i \curlywedge P^{\rightarrow}_i$.

This idea is summarized in Algorithm 8 below. It follows from the definition of $Z^{\leftarrow}_i$ and $Z^{\rightarrow}_i$ that $Z^{\leftarrow}_0 = \mathcal{P}(Q) \setminus \mathcal{P}(F)$ *[step 10]* and $Z^{\rightarrow}_0 = I_0(A)$ *[step 11]*. Thus, we conclude that $P^{\leftarrow}_0 = \{F, Q \setminus F\}$ and $P^{\rightarrow}_0 = \{S_0, Q \setminus S_0\}$. Then, $P^{\leftrightarrow}_0 = P^{\leftarrow}_0 \curlywedge P^{\rightarrow}_0 = \{F, S_0, Q \setminus (F \cup S_0)\}$ *[step 9]*.

Such an initial partition requires to eliminate few extreme cases of given *AFA*s, whose state sets could not be meaningfully used to create a partition $P^{\leftrightarrow}_0$. In case of $Q = F$ or $F = \varnothing$, the emptiness of a given AFA could be decided trivially *[step 1-4]*. If there exists an initial state of A which is final at the same moment, the automaton is obviously non-empty *[step 5-6]*. Finally, if there does not exist a state which is neither initial, nor final, the question is, whether it is possible to leave the set $I_0(A)$ within one step *[step 7-8]*.

In other cases, $P^{\leftrightarrow}_0$ corresponds to a valid partition of Q, which allows us to enter the main loop of Algorithm 8.

First, we check whether the automaton is non-empty using the condition which has been discussed above *[step 13-14]*. If not so, then we create an abstract *AFA* $A^{\alpha}_{P^{\leftrightarrow}_i}$ using the partition $P^{\leftrightarrow}_i$ *[step 15]*.

Next, we compute sets $R^{\leftarrow}_i, R^{\rightarrow}_i$, which should contain all the reachable nodes of $A^{\alpha}_{P^{\leftrightarrow}_i}$, which also belong to an abstraction of $Z^{\leftarrow}_i, Z^{\rightarrow}_i$, respectively. If it is not possible to leave an abstraction of $Z^{\leftarrow}_i$ or $Z^{\rightarrow}_i$ in one step, the given *AFA* is surely empty *[step 18-19]*.

Finally, both sets $Z^{\leftarrow}_i$ and $Z^{\rightarrow}_i$ will be recomputed *[step 20-21]* and the new partition $P^{\leftrightarrow}_{i+1}$ will be created.

Algorithm 8: Forward-like abstract bidirectional algorithm deciding the *AFA* emptiness

Input: AFA $A = (Q, \Sigma, S_0, \delta, F)$
Output: True iff $L(A) = \varnothing$

1 **if** $F = Q$ **then**
2 | **return** False;
3 **if** $F = \varnothing$ **then**
4 | **return** True;
5 **if** $F \cap S_0 \neq \varnothing$ **then**
6 | **return** False;
7 **if** $F \cup S_0 == Q$ **then**
8 | **return** $post_A(I_0(A)) \subseteq I_0(A)$;
9 $P_0^{\leftrightarrow} \leftarrow \{F, S_0, Q \setminus (F \cup S_0)\}$;
10 $Z_0^{\leftarrow} \leftarrow \mathcal{P}(Q) \setminus \mathcal{P}(F)$;
11 $Z_0^{\rightarrow} \leftarrow I_0(A)$;
12 **for** i **in** $\mathbb{N}$ **do**
13 | **if** $Z_i^{\rightarrow} \not\subseteq Z_i^{\leftarrow}$ **then**
14 | | **return** False;
15 | $A^{\alpha}_{P_i^{\leftrightarrow}} \leftarrow (Q^{\alpha}, \Sigma, S_0^{\alpha}, \delta^{\alpha}, F^{\alpha})$;
16 | $R_i^{\leftarrow} \leftarrow \mu x \cdot (I_0(A) \cup post_{A^{\alpha}_{P_i^{\leftrightarrow}}}(x)) \cap \alpha_{P_i^{\leftrightarrow}}(Z_i^{\leftarrow})$;
17 | $R_i^{\rightarrow} \leftarrow \mu x \cdot (I_0(A) \cup post_{A^{\alpha}_{P_i^{\leftrightarrow}}}(x)) \cap \alpha_{P_i^{\leftrightarrow}}(Z_i^{\rightarrow})$;
18 | **if** $post_{A^{\alpha}_{P_i^{\leftrightarrow}}}(R_i^{\leftarrow}) \subseteq \alpha_{P_i^{\leftrightarrow}}(Z_i^{\leftarrow}) \mid\mid post_{A^{\alpha}_{P_i^{\leftrightarrow}}}(R_i^{\rightarrow}) \subseteq \alpha_{P_i^{\leftrightarrow}}(Z_i^{\rightarrow})$ **then**
19 | | **return** True;
20 | $Z_{i+1}^{\leftarrow} \leftarrow \gamma_{P_i^{\leftrightarrow}}(R_i^{\leftarrow}) \cap \widetilde{pre}_A(\gamma_{P_i^{\leftrightarrow}}(R_i^{\leftarrow}))$;
21 | $Z_{i+1}^{\rightarrow} \leftarrow \gamma_{P_i^{\leftrightarrow}}(R_i^{\rightarrow}) \cup post_A(\gamma_{P_i^{\leftrightarrow}}(R_i^{\rightarrow}))$;
22 | $P_{i+1}^{\leftrightarrow} \leftarrow (\curlyvee\{P \in \mathbb{P} \mid \gamma_P^{+}(Z_{i+1}^{\leftarrow}) = Z_{i+1}^{\leftarrow}\}) \curlywedge (\curlyvee\{P \in \mathbb{P} \mid \gamma_P^{+}(Z_{i+1}^{\rightarrow}) = Z_{i+1}^{\rightarrow}\})$;
23 **end**

5.3 Backward-like Abstract Bidirectional Algorithm

Analogously, it is possible to define the dual algorithm to Algorithm 8. Due to the fact that both algorithms share several ideas, the backward-like abstract bidirectional algorithm won't be described in detail.

Algorithm 9: Backward-like abstract bidirectional algorithm deciding the *AFA* emptiness

---

Input: AFA $A = (Q, \Sigma, S_0, \delta, F)$
Output: True iff $L(A) = \varnothing$

1 **if** $F = Q$ **then**
2 | **return** False;
3 **if** $F = \varnothing$ **then**
4 | **return** True;
5 **if** $F \cap S_0 \neq \varnothing$ **then**
6 | **return** False;
7 **if** $F \cup S_0 == Q$ **then**
8 | **return** $post_A(I_0(A)) \subseteq I_0(A)$;
9 $P_0^{\leftrightarrow} \leftarrow \{F, S_0, Q \setminus (F \cup S_0)\}$;
10 $Z_0^{\leftarrow} \leftarrow \mathcal{P}(Q) \setminus I_0(A)$;
11 $Z_0^{\rightarrow} \leftarrow \mathcal{P}(F)$;
12 **for** i **in** $\mathbb{N}$ **do**
13 | **if** $Z_i^{\rightarrow} \not\subseteq Z_i^{\leftarrow}$ **then**
14 | | **return** False;
15 | $A^{\alpha}_{P_i^{\leftrightarrow}} \leftarrow (Q^{\alpha}, \Sigma, S_0^{\alpha}, \delta^{\alpha}, F^{\alpha})$;
16 | $R_i^{\leftarrow} \leftarrow \mu x \cdot (\mathcal{P}(F) \cup pre_{A^{\alpha}_{P_i^{\leftrightarrow}}}(x)) \cap \alpha_{P_i^{\leftrightarrow}}(Z_i^{\leftarrow})$;
17 | $R_i^{\rightarrow} \leftarrow \mu x \cdot (\mathcal{P}(F) \cup pre_{A^{\alpha}_{P_i^{\leftrightarrow}}}(x)) \cap \alpha_{P_i^{\leftrightarrow}}(Z_i^{\rightarrow})$;
18 | **if** $pre_{A^{\alpha}_{P_i^{\leftrightarrow}}}(R_i^{\leftarrow}) \subseteq \alpha_{P_i^{\leftrightarrow}}(Z_i^{\leftarrow}) \mid\mid pre_{A^{\alpha}_{P_i^{\leftrightarrow}}}(R_i^{\rightarrow}) \subseteq \alpha_{P_i^{\leftrightarrow}}(Z_i^{\rightarrow})$ **then**
19 | | **return** True;
20 | $Z_{i+1}^{\leftarrow} \leftarrow \gamma_{P_i^{\leftrightarrow}}(R_i^{\leftarrow}) \cap \widetilde{post}_A(\gamma_{P_i^{\leftrightarrow}}(R_i^{\leftarrow}))$;
21 | $Z_{i+1}^{\rightarrow} \leftarrow \gamma_{P_i^{\leftrightarrow}}(R_i^{\rightarrow}) \cup pre_A(\gamma_{P_i^{\leftrightarrow}}(R_i^{\rightarrow}))$;
22 | $P_{i+1}^{\leftrightarrow} \leftarrow (\curlyvee\{P \in \mathbb{P} \mid \gamma_P^{+}(Z_{i+1}^{\leftarrow}) = Z_{i+1}^{\leftarrow}\}) \curlywedge (\curlyvee\{P \in \mathbb{P} \mid \gamma_P^{+}(Z_{i+1}^{\rightarrow}) = Z_{i+1}^{\rightarrow}\})$;
23 **end**

5.4 Properties of Forward-like and Backward-like Algorithms

Although the proposed algorithms iteratively build two different sets, we are still able to work with only one abstract *AFA* within each iteration of their main loops. They also require to compute a fixed point of $post_{A^{\alpha}_{P_i^{\leftrightarrow}}}$ or $pre_{A^{\alpha}_{P_i^{\leftrightarrow}}}$ only once within each iteration, because both $R_i^{\leftarrow}$ and $R_i^{\rightarrow}$ are calculated using the same fixed point of a predicate transformer. This implies that the forward-like and backward-like algorithms can reuse previously computed results and thus save computational time. However, both forward-like and backward-like algorithms always use less coarse partition to create a corresponding abstract *AFA* than the naive approach because they require to represent two different sets in the same abstract domain, which can possibly decrease the efficiency.

Chapter 6

Design and Implementation

In this chapter, we will describe the data structures which are used in the implementation of algorithms presented in previous parts of the thesis. First, we will discuss the representation of an *AFA* and corresponding data structures which facilitate evaluation of predicate transformers.

Next, the inner representation of a closed set will be described. Due to the fact that it is crucial to manipulate effectively with closed sets, we will also focus on the implementation of set operations over them.

Subsequently, we will pay attention to the data structure which represents a partition of a set and to the possibility of mapping between an concrete and abstract domain of an *AFA* using this data structure.

All the presented data structures and algorithms were implemented in the `C++` language using the standard libraries and containers (namely `std::set, std::vector, std::list, std::map`) with respect to the object-oriented paradigms.

6.1 Alternating Automata

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA*. Suppose that $\mathbb{N}_n \triangleq \{i \in \mathbb{N} \mid i \leq n\}$ for any $n \in \mathbb{N}$.

Due to the fact that the only meaning of names of states is the possibility to distinguish them, we can easily perform a substitution using the bijective maps $\theta : Q \to \mathbb{N}_{|Q|-1}$ and $\Theta : \mathcal{P}(Q) \to \mathcal{P}(\mathbb{N}_{|Q|-1})$, where $\Theta(X) = \bigcup_{x \in X}\{\theta(x)\}$ without any loss of information.

Thus, it is sufficient to create an inner representation for $A_\Theta = (\Theta(Q), \Sigma, \Theta(S_0), \delta_\Theta, \Theta(F))$, where $\delta_\Theta = \{((\theta(q), a), \bigvee_{i=0}^{m} \bigwedge_{j=0}^{n} \theta(\varphi_{i,j})) \mid ((q, a), \bigvee_{i=0}^{m} \bigwedge_{j=0}^{n} \varphi_{i,j}) \in \delta\}$. Next, we will discuss an inner representation of each part of A_Θ.

Transititon relation δ_Θ. It will be represented as follows. Let $(q, a, \phi) \in \delta_\Theta$ be a transition of A_Θ. Without loss of generality, it is possible to say that ϕ is in a disjunctive normal form, which means that $\phi = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} \varphi_{i,j}$. Then, we can represent ϕ using the set $\phi_\Theta = \bigcup_{i=1}^{m}\{\bigcup_{j=1}^{n}\{\varphi_{i,j}\}\}$, which allows us to create a `Transition` structure which holds the triplet (q, a, ϕ_Θ).

Then, we can create a `TransitionList` as a list of `Transitions`, such that each transition in a transition list shares the former state q with each other, while the symbols on transition always differ.

Finally, the data structure `TransitionRelation` consists of `TransitionLists` such that the former states across each transition list differ. The `TransitionRelation` is represented as a vector, which means that there is no need to explicitly hold the information about the name of each state since the name corresponds to an index of the vector.

State set $\Theta(Q)$. Since $\Theta(Q)$ corresponds to indices of a vector of length $|Q|$, we can simply leave the set $\Theta(Q)$ implicit.

Alphabet Σ. Analogously, there is no need to store Σ in our inner representation of A_Θ explicitly since the information about used symbols is already part of the relation δ_Θ.

Sets of initial and final states $\Theta(S_0), \Theta(F)$. Both sets $\Theta(S_0)$ and $\Theta(F)$ will be conventionally represented as sets of states.

Example. Suppose that A is an AFA defined in Example 3.1.1 and depicted in Figure 3.1. The diagram shown in Figure 6.1 corresponds to an inner representation of δ_Θ.

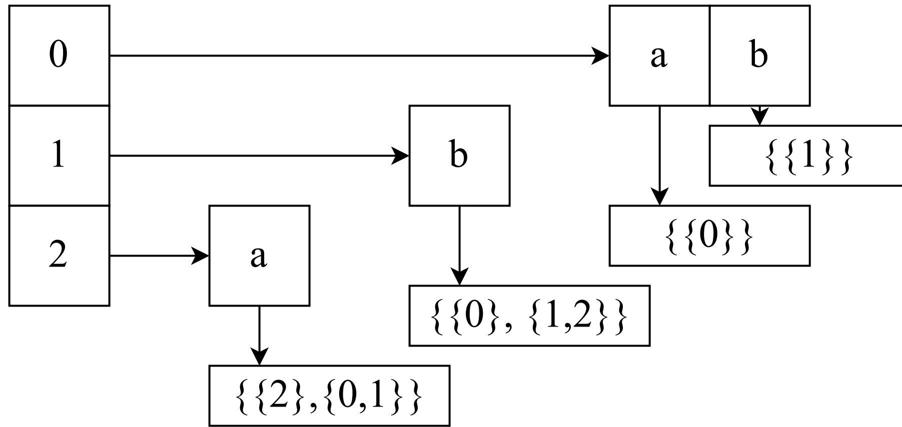

Figure 6.1: Inner representation of a transition relation of an AFA A

6.2 Closed Sets

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA and $X \subseteq \mathcal{P}(Q)$ is an upward-closed or downward-closed set.

Due to the fact that every state set consist of a finite sequence of natural numbers starting with 0, we can say that $Q = \mathbb{N}_{|Q|-1}$. This implies that there is no need to represent neither $\mathcal{P}(Q)$ nor Q as a carrier of X explicitly, we can simply store a value $|Q|-1$ in our data structure as a maximal element of a state set and leave the rest of states implicit.

Next, we need to know whether a closed set is upward-closed or downward-closed.

Due to the relation between closed sets and sets of minimal or maximal elements, it is sufficient to represent each closed set with its corresponding antichain.

These three information unambiguously describe every possible closed set of nodes of a given AFA.

In what follows, we will describe commonly used set relations and set operations in context of closed sets and an efficient way to evaluate them. Let $X, Y \subseteq \mathcal{P}(Q)$ be upward-closed sets and $x \in \mathcal{P}(Q)$.

Membership. We can say that $x \in \uparrow X \iff \exists x_0 \in \lfloor \uparrow X \rfloor : x_0 \subseteq x$. This implies that it is sufficient to simply test a set inclusion between x and each element of antichain corresponding to $\uparrow X$.

Inclusion. Since $\uparrow Y \subseteq \uparrow X \iff \forall y_0 \in \lfloor \uparrow Y \rfloor : y_0 \in \uparrow X$, we can simply repeatedly perform a membership test described above.

Union. The equation $\uparrow X \cup \uparrow Y = \uparrow \lfloor \lfloor \uparrow X \rfloor \cup \lfloor \uparrow Y \rfloor \rfloor$ holds. We can thus simply create an union of antichains corresponding to given closed sets.

Intersection. We know that $\uparrow X \cap \uparrow Y = \uparrow \{x_0 \cup y_0 \mid (x_0, y_0) \in \lfloor X \rfloor \times \lfloor Y \rfloor\}$. It is thus sufficient to compute an union of every pair of elements of antichains corresponding to given closed sets.

Complement. To express an complement of a set unambiguously, let us suppose that $\overline{X}^Q \triangleq Q \setminus X$, where the superscript emphasises the context. For a single element $x \in \mathcal{P}(Q)$, we can say that $\overline{\uparrow\{x\}}^{\mathcal{P}(Q)} = \downarrow\{\bigcup_{x_0 \in x} \overline{\{x_0\}}^Q\}$. Since $\uparrow X = \bigcup_{x \in \lfloor X \rfloor} \uparrow\{x\}$, we conclude that

$$\overline{\uparrow X}^{\mathcal{P}(Q)} = \downarrow \bigcap_{x \in \lfloor X \rfloor} \Big\{ \bigcup_{x_0 \in x} \overline{\{x_0\}}^Q \Big\}.$$

This implies that the complement of a closed set could be easily computed using ideas of intersection and union mentioned above and using a compelement of Q. It is easy to see that the complement of an upward-closed set will always be downward-closed and vice versa.

Notice that all the procedures explained above could be used similarly in context of downward-closed sets. However, it is not possible to combine upward-closed and downward-closed sets while performing presented binary set operations.

6.3 Predicate Transformers

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an AFA. In what follows, we will describe the evaluation of predicate transformers in context of our representation of an AFA.

$\mathbf{Post_A}$. Let $x \in \mathcal{P}(Q)$. To compute $Post_A(x)$, it is necessary to perform $|x|$ accesses to the `TransitionRelation` vector of A (for each $x_0 \in x$), use the results to create $|x|$ corresponding upward-closed sets. The result is an intersection of these upward-closed sets, which is also an upward-closed set.

Each result of $Post_A$ is immediately stored to the cache to avoid its potential recomputation.

$\mathbf{post_A}$. Let $X \subseteq \mathcal{P}(Q)$. Then, $post_A(X)$ is computed as $\bigcup_{x \in \lfloor X \rfloor} Post_A(x)$, which is an upward-closed set.

$\widetilde{\mathbf{post}}_{\mathbf{A}}$**.** Let $X \subseteq \mathcal{P}(Q)$. Then, $\widetilde{post}_A(X)$ is computed as $\overline{post_A(\overline{X})}$, which is a downward-closed set.

$\mathbf{Pre_A}$**.** To be able to efficiently compute the predicate transfomer Pre_A in our implementation, we need to somehow represent an *inverse transition relation* δ^{-1} of A. Due to the fact that δ^{-1} is not an explicit part of A and it follows from Q, Σ and δ implicitly, it is necessary to compute δ^{-1} using these components of A.

Note that we cannot simply create $\delta^{-1} : B^+(Q) \times \Sigma \to Q$ by switching first and third element of each $(q, a, \phi) \in \delta$, since δ is not injective. The other possibility is to create a function $\delta^{-1} : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(\mathcal{P}(Q))$ such that $\delta^{-1}(N_1, a) = \{N_0 \subseteq Q \mid \exists (N_0, a, N_1) \in E(A)\}$ for each $N_1 \subseteq Q$. Then, we can simply say that $Pre_A(N) = \bigcup_{a\in\Sigma} \delta^{-1}(N, a)$. However, this straightforward method leads to the necessity of storing an exponentially many subsets of Q to the memory, which we require to avoid.

Thus, we propose the following definition of $\delta^{-1} : Q \times \Sigma \to \mathcal{P}(\mathcal{P}(Q) \times \mathcal{P}(Q))$ and the data structure which represents it.

Let $\delta^{-1}(q, a) = \{(N_1, S_1), (N_2, S_2)\}$. Note that each element of the result set is a tuple. In what follows, we will call first elements of such tuples *result nodes* and second elements will be called *sharing lists*. The sharing list always contains a former element q which was on input of the function. It means that $q \in S_1$ and $q \in S_2$. Furthemore, suppose that $s \in S_1$. Then, we can conclude that $(N_1, S_1) \in \delta^{-1}(s, a)$, which means that both q, s *share* the same tuple (N_1, S_1) within their output if a is used as an input symbol.

Suppose that $n \in N_1$. Then, we can conclude that there exists a triplet $(n, a, \phi) \in \delta$, such that $S_1 \in \lfloor [\![\phi]\!] \rfloor$. In other words, each element of a result node N_1 could be used to perform a transition via a symbol a to the node S_1.

Then, we can alternatively define Pre_A using δ^{-1} as follows:

$$Pre_A(N_1) = \downarrow \bigcup_{a\in\Sigma} \left\{ \bigcup_{n\in N_1} \left\{ N_0 \;\middle|\; (N_0, S_0) \in \delta^{-1}(n, a) \wedge S_0 \subseteq N_1 \right\} \right\}$$

Example. Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA* defined in Example 3.1.1 and depicted in Figure 3.1. The graphical representation of the corresponding function δ^{-1} of A is shown in Figure 6.2. The first row represents a set Q and the second row depicts an alphabet for each $q \in Q$. The boxes below represent tuples $(N, S) \in \mathcal{P}(Q) \times \mathcal{P}(Q)$. The upper element corresponds to a result node, while the lower states for a sharing list.

Let us thoroughly describe a computation of $Pre_A(\{0, 1\})$.

To compute $Pre_A(\{0, 1\})$, we firstly need to evaluate both $\delta^{-1}(0, a)$ and $\delta^{-1}(1, a)$. It is obvious from Figure 6.2 that $\delta^{-1}(0, a) = \{(\{0\}, \{0\}), (\{2\}, \{0, 1\})\}$ and $\delta^{-1}(1, a) = \{(\{2\}, \{0, 1\})\}$. Since both $(\{0\}, \{0\}), (\{2\}, \{0, 1\})$ are tuples whose sharing list is a subset of the input $\{0, 1\}$, we create the union $\{0\} \cup \{2\} = \{0, 2\}$ of corresponding result nodes. Analogously, we continue with the symbol b. Notice that $\delta^{-1}(0, b) = \{(\{1\}, \{0\})\}$ and $\delta^{-1}(1, b) = \{(\{0\}, \{1\}), (\{1\}, \{1, 2\})\}$. In this case, only $(\{1\}, \{0\})$ and $(\{0\}, \{1\})$ are found tuples whose sharing list is a subset of the input $\{0, 1\}$, so we ignore the other one. We should thus remember the union $\{0, 1\}$ of the result nodes $\{0\}$ and $\{1\}$.

Thus, $pre_A(\{0,1\}) = \downarrow\{\{0,2\},\{0,1\}\}$. We can simply check the result using the diagram depicted in Figure 6.1 and convince ourselves that for each $N \in\downarrow\{\{0,2\},\{0,1\}\}$, the relation $\{0,1\} \in Post_A(N)$ holds, while for any other node this relation does not hold.

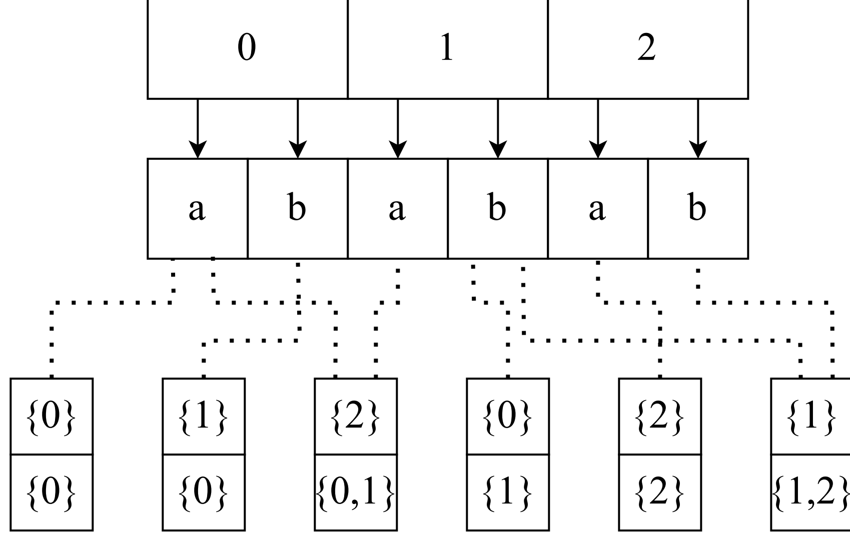

Figure 6.2: Inner representation of an inverse transition function δ^{-1} of an *AFA* A

$\mathbf{pre_A}$. Let $X \subseteq \mathcal{P}(Q)$. Then, $pre_A(X)$ is computed as $\bigcup_{x \in \lfloor X \rfloor} Pre_A(x)$, which is a downward-closed set.

$\widetilde{\mathbf{pre}}_{\mathbf{A}}$. Let $X \subseteq \mathcal{P}(Q)$. Then, $\widetilde{pre}_A(X)$ is computed as $\overline{pre_A(\overline{X})}$, which is an upward-closed set.

6.4 Partitions

Let $A = (Q, \Sigma, S_0, \delta, F)$ be an *AFA* and P is a partition of Q. According to the definition of a partition, $P \subseteq \mathcal{P}(Q)$. Due to the fact that the partition corresponds to a state set of an abstract *AFA*, we want to represent each class of P as a natural number $n \in \mathbb{N}_{|P|-1}$ rather than work with P explicitly.

Therefore, it is necessary to store the value $|P|-1$ in our inner representation of P and create maps $P_Q : Q \to P$, $f : \mathbb{N}_{|P|-1} \to P$, $f^{-1} : P \to \mathbb{N}_{|P|-1}$ to facilitate conversions between concrete and abstract domains of A.

Chapter 7

Experimental Comparison of Algorithms Deciding AFA Emptiness

In this chapter, we will present results of experimental evaluation of concrete, abstract and abstract bidirectional algorithms over various *AFA*s. Measured values will be compared and used to describe few classes of *AFA*s whose emptiness could be efficiently decided using each algorithm.

First, we introduce the *Tabakov-Vardi model* for generating random nondeterministic finite automata and Büchi automata using uniform probability distribution [8, 3]. Subsequently, we describe proposed extension of Tabakov-Vardi model which will be able to randomly generate alternating automata.

Next, we discuss the significant efficiency bottlenecks of abstract algorithms and we propose several optimizations which reduce the cost of studied algorithms. We also experimentally evaluate the optimized algorithms over randomly generated *AFA*s to find out whether proposed modifications accelerate the computation.

At the end of the chapter, we consider a few problems which are decidable by performing an *AFA* emptiness test and we try to find out whether the abstract algorithms give us better results in context of these problems.

All the experiments were performed on a machine with the *Intel Xeon E5-2630 @ 2.60Hz* processor and 32 GB RAM.

7.1 Tabakov-Vardi Model

Definition 49 *Let* $\mathbb{TV} \triangleq (\mathbb{N} \setminus \{0,1\}) \times \mathbb{R}^+ \times (0;1)$*. In what follows, a triplet* $(n,d,f) \in \mathbb{TV}$*, where* n *is a* ***state-set cardinality****,* d *is a* ***transition density*** *and* f *is a* ***final states density****, will be called a* ***generator of Tabakov-Vardi random automaton****. A class* $\mathbb{TV}$ *thus contains all possible generators of Tabakov-Vardi random automata.*

Suppose that $G = (n, d, f) \in \mathbb{TV}$. We want to generate random *NFA* using these parameters. Let us say that $A = (Q, \Sigma, S_0, R, F)$ is an *NFA*, which was randomly generated by G. Since a state-set cardinality n corresponds to a fixed positive number, we simply state that $Q = \mathbb{N}_{n-1}$. Due to the fact that no parameter of G describes an alphabet and a set of initial states of A, we define $\Sigma = \{0, 1\}$ and $S_0 = \{0\}$ for all automata generated by a generator from $\mathbb{TV}$.

The final states density f describes a ratio between the number of final states and the amount of all the states. This implies that $F \subseteq Q \setminus S_0$ is a randomly chosen set, such that $|F| = \lfloor n \cdot f \rfloor$. Due to the fact that the emptiness test of an *NFA* is trivially decidable if $S_0 \cap F \neq \varnothing$, the initial state 0 is not allowed to be randomly generated as a final state.

The remaining part of G, which is denoted by d, approximately expresses a ratio between number of transitions which use one symbol $a \in \Sigma$ and amount of all states of A. We can thus write $|R| \approx |\Sigma| \cdot n \cdot d$. The transitions are generated as follows. For each $a \in \Sigma$, $n \cdot d$ tuples $(q_1, q_2) \subseteq Q \times Q$ are randomly generated and stored in a set D_a. Due to the fact that a set cannot contain duplicate elements, the cardinality $|D_a|$ could be possibly less than $n \cdot d$. Finally, the transition relation is chosen as $\bigcup_{a \in \Sigma} \{(q_1, a, q_2) \mid (q_1, q_2) \in D_a\}$.

7.2 Extended Tabakov-Vardi Model

Due to the fact that the Tabakov-Vardi model enables us to generate only *NFA*s and Büchi automata, we propose an extension of the introduced model which is able to generate *AFA*s and also influence the probability of increasing the number of reachable nodes in an *AFA*.

Definition 50 *Let* $\mathbb{ETV} \triangleq (\mathbb{N} \setminus \{0, 1\}) \times \mathbb{R}^+ \times (0; 1) \times \langle 0; 1\rangle \times \langle 0; 1)$. *In what follows, every quintuple* $(n, d, f, p, c) \in \mathbb{ETV}$*, where* n *is a* ***state set cardinality****,* d *is a* ***transition density****,* f *stands for a* ***final states density****,* p *corresponds to a* ***revisition probability*** *and* c *is an* ***alternation probability****, will be called a* ***generator of Extended Tabakov-Vardi random automaton****. Thus,* $\mathbb{ETV}$ *is a class of all possible generators of Extended Tabakov-Vardi random automata.*

Suppose that $G = (n, d, f, p, c) \in \mathbb{ETV}$ and $A = (Q, \Sigma, S_0, \delta, F)$ is an *AFA* randomly generated by G. All three values n, d, f share the same meaning with a former Tabakov-Vardi model. Analogously, $Q = \mathbb{N}_{n-1}, \Sigma = \{0, 1\}, S_0 = \{0\}$ and $F \subseteq Q \setminus S_0$, where $|F| = \lfloor n \cdot f \rfloor$.

The main difference between these two models lies in the random choice of a transition relation. We have already seen that the destination of an alternating transition could be represented as a node of A. Thus, for each $a \in \Sigma$, we require to generate sets $D_a \subseteq Q \times \mathcal{P}(Q)$ rather than $D_a \subseteq Q \times Q$. The rate of alternation is described by the value c. Suppose that $(q, N) \in D_a$. It is always guaranteed that the generated successor of q contains at least one element. The probability of generating k additional elements equals to c^k, which means that the probability of generating another element decreases geometrically. Note that in case of $c = 0$, G in fact always generates an *NFA*, since there cannot be any alternating transition.

Let $(q, a, \phi) \in \delta$ be a transition randomly generated by G and $N = \lfloor [\![\phi]\!] \rfloor$. The remaining element of G, which is denoted by p and corresponds to a revisition probability, reduces

a size of the set of states which could be chosen as q. Since δ is generated iteratively by constructing sets D_a, where $a \in \Sigma$, we can say that $D_a^n \subseteq D_a$ is the result of generating D_a in n^{th} iteration and $D_a^0 = \varnothing$. Then, when generating an element (q_n, a, N_n) in the nth iteration, we uniformly choose $q_n \in Q$ with probability of $1-p$. However, with probability of p, q_n is generated from the set $\{q \in N_i \mid \exists(q_i, a, N_i) \in D_a^{n-1}\} \cup S_0 \subseteq Q$, which means that q has been already generated or it is an initial state. Note that this does not necessarily guarantees us that with probability of p, q is a reachable state because we also allow generating alternating transitions. However, in the special case of $p = 1$ and $c = 0$, each state which is a destination of a transition is always reachable because A is a *NFA*.

7.3 Initial Observation of Randomly Generated AFAs Behaviour

In this section, we try to generate plenty of *AFA*s using the extended Tabakov-Vardi model with various parameters and summarize time of evaluating our algorithms over them.

Let $T \subseteq \mathbb{ETV}$ be a finite set of extended Tabakov-Vardi model generators. Concretely, we consider all quintuples $(n, d, f, p, c) \in \mathbb{EVT}$, such that $n \in \{10, 20, 30, 40\}$, $f = 0.1$, $d \in \{1, 1.2, 1.4, 1.6, 1.8, 2\}$, $p \in \{0, 0.25, 0.5, 0.75, 1\}$ and $c \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For each generator $G \in T$, we will generate 20 separate *AFA*s and evaluate previously presented algorithms over them. Altogether, we will work with 12 000 *AFA*s generated using various parameters of extended Tabakov-Vardi model.

The Table 7.1 summarizes measured time of performing presented algorithms over these *AFA*s. Let us recall that *A. Forw.* states for the Algorithm 6, *A. Back.* is the Algorithm 7, *C. Forw.* corresponds to the Algorithm 4, *C. Back.* refers to the Algorithm 5 and finally, *Bi. Forw.* and *Bi. Back.* correspond to newly proposed Algorithms 8, 9, respectively.

The first column ephasises the cardinality of state sets of *AFA*s, while the others correspond to measured time in seconds. The symbol $\varnothing$ states for a arithmetical mean and $\widetilde{t}$ represents a median.

	A. Forw.		A. Back.		C. Forw.		C. Back.		Bi. Forw.		Bi. Back.	
$\lvert Q \rvert$	$\varnothing$	$\widetilde{t}$	$\varnothing$	$\widetilde{t}$	$\varnothing$	$\widetilde{t}$	$\varnothing$	$\widetilde{t}$	$\varnothing$	$\widetilde{t}$	$\varnothing$	$\widetilde{t}$
10	.014	.010	.008	.006	.002	.001	.001	.001	.007	.004	.006	.004
20	.328	.059	.249	.013	.026	.003	.003	.002	.072	.016	.163	.013
30	4.343	.267	2.025	.027	.031	.004	.006	.003	.209	.050	.694	.036
40	15.234	.880	4.768	.045	.023	.005	.007	.004	.455	.129	1.243	.089

Table 7.1: Duration of evaluation the algorithms over 12 000 various *AFA*s in seconds.

We have found out that in case of total random *AFA*s, we do not gain any advantage while using abstract algorithms in general. Let us focus on the behaviour of abstract algorithms and inspect which steps of them represent the most significant efficiency bottleneck.

Let us reuse those generators $T \subseteq \mathbb{ETV}$ presented above which generate *AFAs* with 30 states. Suppose that for each generator $G \in T$, which satisfies this condition, we generate

20 AFAs, which means that we work with 3 000 AFAs in total. All four algorithms which work over an abstract domain will be evaluated over these AFAs.

The bar charts depicted in Figure 7.1 summarize average computational time of some lines of Algorithms 6, 7, 8 and 9. Since forward and backward algorithms act similarly, we show them within the same chart. Note that we did not covered all lines of these algorithms because most of them deal with trivial operations. To create these charts, we have computed average time of computation with no respect to number of executed iterations of these algorithms. Notice that the y-axis is shown in logarithmic scale.

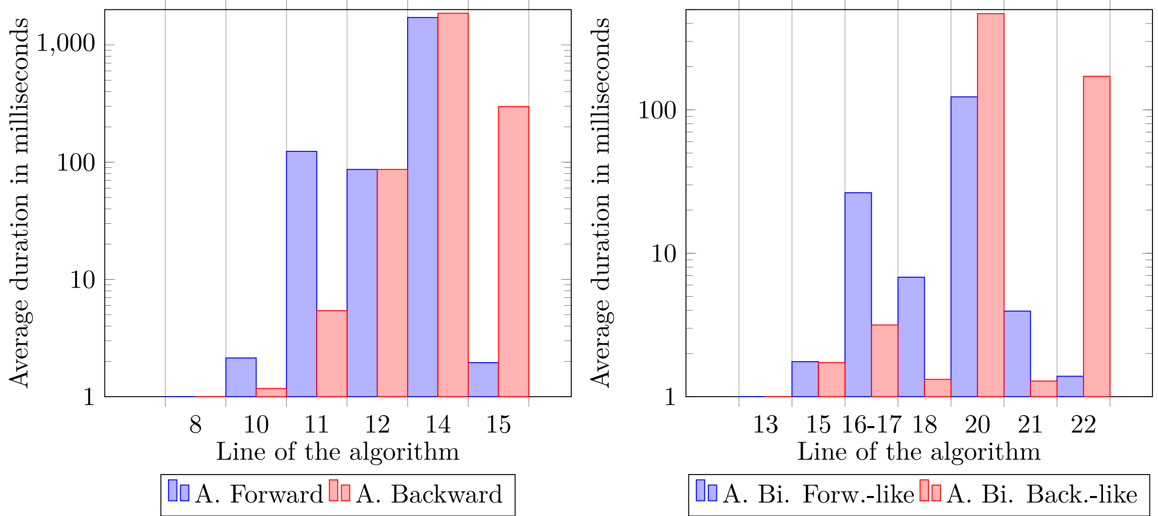

Figure 7.1: Average execution time of individual lines of presented algorithms over 3 000 random AFAs

We have observed that the lines number 14 or former abstract algorithms (computation of $\gamma_{P_i}(R_i) \cap \widetilde{pre}_A(\gamma_{P_i}(R_i))$ and $\gamma_{P_i}(R_i) \cap \widetilde{post}_A(\gamma_{P_i}(R_i))$) and lines 20 of proposed bidirectional algorithms (computation of $\gamma_{P_i^{\leftrightarrow}}(R_i) \cap \widetilde{pre}_A(\gamma_{P_i^{\leftrightarrow}}(R_i))$ and $\gamma_{P_i^{\leftrightarrow}}(R_i) \cap \widetilde{post}_A(\gamma_{P_i^{\leftrightarrow}}(R_i))$) correspond to the least effective ones because their execution reuqieres significantly more computational time than the others.

Let us recall that $\widetilde{pre}_A(X) = \overline{pre_A(\overline{X})}$ and $\widetilde{post}_A(X) = \overline{post_A(\overline{X})}$. This implies that the discussed lines of our algorithms consist of five various operations, concretely conversion to a concrete domain $X_1 = \gamma_{P_i}(R_i)$, a complementation $X_2 = \overline{X_1}$, an evaluating of predicate transformer $X_3 = pre_A(X_2)$ (or $X_3 = post_A(X_2)$), another complementation $X_4 = \overline{X_3}$ and an intersection $X_5 = X_1 \cap X_4$.

To understand better which of these operations requires most computational time, we have created additional bar charts depicted in Figure 7.2 which express this information. The charts summarize the behaviour of the same 3 000 AFAs mentioned above while executing discussed operations.

We have observed that the intersection of two closed sets in the concrete domain and the complementation of a closed set correspond to the operations which takes the most computational time. Notice also the column 15 in the Firuge 7.1 on left and the column 22 on right. These lines create a new partition by computing X-neighbours. The considerable difference between behaviour of forward and backward algorithms is caused by properties of representability of a closed set in an abstract domain mentioned in Section 4.4. In case

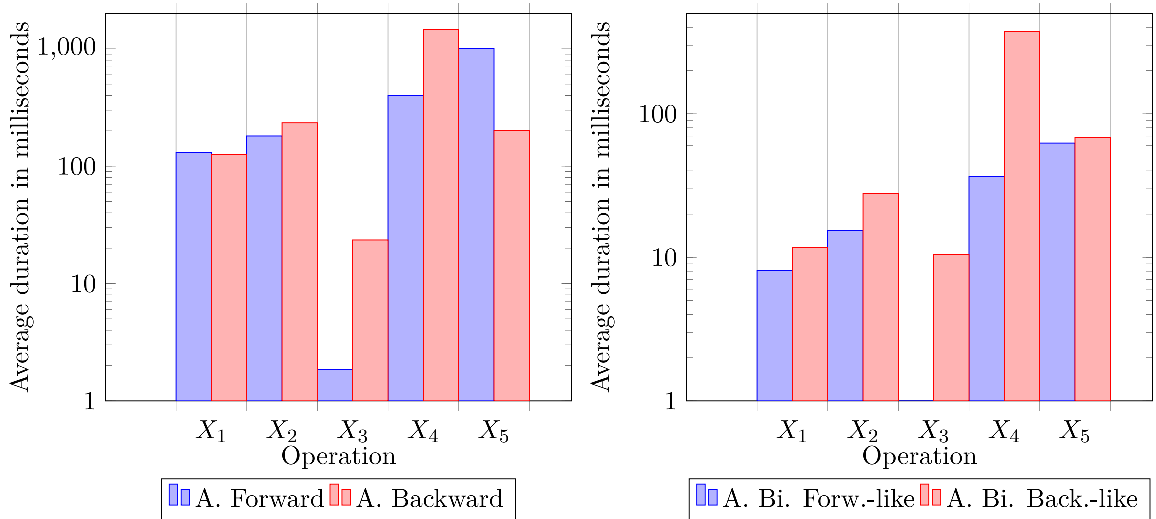

Figure 7.2: Average execution time of some operations over 3 000 random *AFA*s

of downward-closed sets, we need to perform two more complementation of a closed set, which significantly disadvantages both abstract backward and the abstract bidirectional backward-like algorithms.

Due to the fact that the concrete algorithms compute the complement of a closed set only once and they also compute an intersection of closed sets only over smaller domain when computing $Pre_A(.)$ or $Post(.)$, it is understandable that their evaluation over 12 000 various *AFA*s gave us significantly better results. To reduce the influence of computing an intersection and a complement, we can consider several strategies described in following paragraphs in detail.

Implement intersection and complementation more effectively. Both implemented algorithms for computing an intersection and a complementation of a closed set explained in Chapter 6 already use all the benefits of the theory of closed sets and antichains presented in Chapter 3. The author of this thesis could not find any way to improve these algorithms of intersection and complementation of closed sets to decrease the differences between concrete and abstract language emptiness tests.

Modify the algorithms to avoid the necessity of computing intersection and complementation so often. Several attempts of changing the former abstract algorithms to achieve this goal had been done by the author of this thesis. Since we are able to derive

$$
\begin{aligned}
&\gamma_{P_i}(R_i) \cap \widetilde{pre}_A(\gamma_{P_i}(R_i)) \\
&= \gamma_{P_i}(R_i) \cap \overline{pre_A(\overline{\gamma_{P_i}(R_i)})} \\
&= \overline{\overline{\gamma_{P_i}(R_i) \cap \overline{pre_A(\overline{\gamma_{P_i}(R_i)})}}} \\
&= \overline{\overline{\gamma_{P_i}(R_i)} \cup \overline{\overline{pre_A(\overline{\gamma_{P_i}(R_i)})}}} \\
&= \overline{\overline{\gamma_{P_i}(R_i)} \cup pre_A(\overline{\gamma_{P_i}(R_i)})},
\end{aligned}
$$

we can possibly avoid the computation of an intersection and still perform a complementation only twice due to the fact that the subresult $\overline{\gamma_{P_i}(R_i)}$ can be reused. However, the other complementation is executed over a larger set which can possible cause another deceleration. Analogously, we can conclude that $\gamma_{P_i}(R_i) \cap \widetilde{post}_A(\gamma_{P_i}(R_i)) = \overline{\overline{\gamma_{P_i}(R_i)} \cup post_A(\overline{\gamma_{P_i}(R_i)})}$. In what follows, we will refer to this modification as the first optimization of abstract algorithms.

Next, we can try to completely change the semantics of discussed lines to avoid executing intersection over large closed sets, yet preserve the correctness of studied algorithms. The intersection in the expression $\gamma_{P_i}(R_i) \cap \widetilde{pre}_A(\gamma_{P_i}(R_i))$ decreases the size of newly computed Z_i by excluding such nodes from $\widetilde{pre}_A(\gamma_{P_i}(R_i))$ which do not belong to $\gamma_{P_i}(R_i)$. Suppose that we omit the intersection and state that $Z_i = \widetilde{pre}_A(\gamma_{P_i}(R_i))$. Then, Z_i still corresponds to an over-approximation of nodes which cannot reach an final node in i or less steps. However, we have lost the crucial property of former Z_i, which says that the sequence of $Z_i, Z_{i+1}, \ldots$ is strictly decreasing and it therefore ensures us that in case of $L(A) = \varnothing$, the algorithm always terminates. To elude entering an infinite loop, we can store each newly computed Z_i in a cache and check whether we have already seen the same set of nodes before. If so, we have detected entering an infinity loop and we are thus able to terminate with result of $L(A) = \varnothing$. Note that we are working in a finite domain, which means that the infinite loop caused by omitting the intersection will be always detected using this procedure.

This idea allows us to avoid computationally expensive intersections of closed sets. However, it might cause executing more main loops of the algorithm since the newly computed Z_i is not as precise as before. In addition, we are now required to store an information which maximal size is bounded by $|Q_{\lfloor . \rfloor}|$ to the memory.

Analogously, we can state that $Z_i = \widetilde{post}_A(\gamma_{P_i}(R_i))$. In what follows, we will refer to this modification as the second optimization of abstract algorithms.

To find out whether these proposed optimizations significantly reduce the cost of studied algorithms, we use a set of generators $T \subseteq \mathbb{ETV}$, where $s \in \{15, 20, 25, 30, 35\}, f = 0.1, d \in \{1, 1.5, 2, 2.5, 3\}, c \in \{0.1, 0.3, 0.5\}, p \in \{0, 0.5, 1\}$. For each $G \in T$, we generate 20 *AFA*s, which means that we use 4 500 automata in total. We execute both abstract forward and abstract bidirectional forward-like algorithms over these *AFA*s using their former definition and both presented optimizations. We also perform a concrete forward test over these *AFA*s to be able to determine whether our optimizations work better than it. The measured results are depicted in Figure 7.3.

We have found out that not even the optimizations of former abstract algorithms cannot beat the concrete one on average time of execution over random *AFA*s. However, when

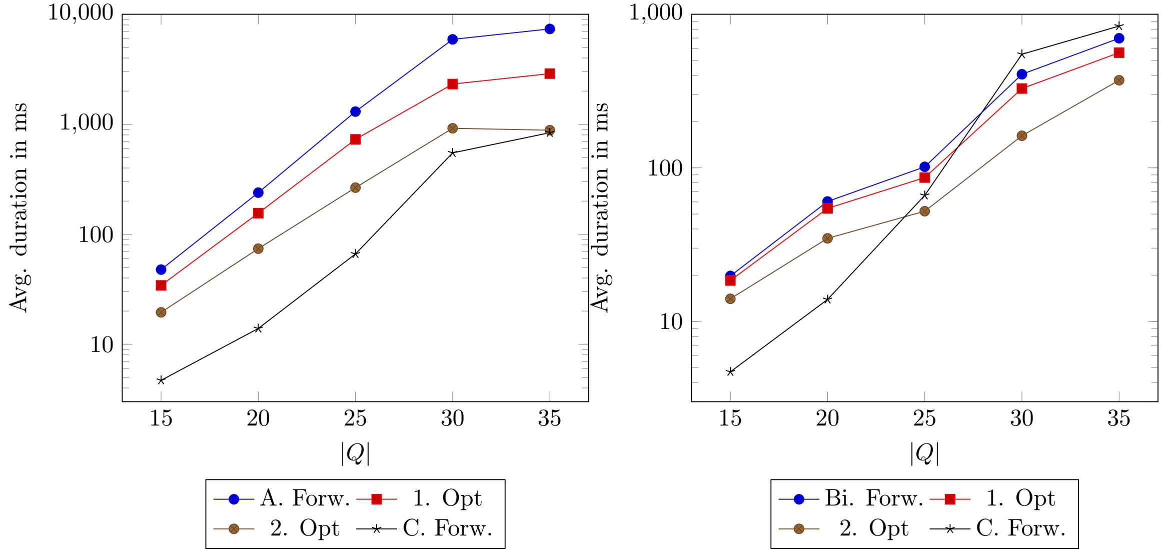

Figure 7.3: Comparison of studied algorithms and their optimizations over 4 500 random *AFA*s

using these optimizations in context of proposed bidirectional algorithms, the cost of the emptiness test working in an abstract domain happens to be lower than the test operating in a concrete domain.

Look for a class of *AFA*s which disadvantages the concrete algorithms We have found out that the computation of intersection and complementation of closed sets correspond to the most costly operations. To illustrate the influence of closed sets cardinality on the computation time, we have measured the average and maximal sizes of sets computed during the process of abstract language emptiness tests over the same 3 000 *AFA*s mentioned in the paragraphs above.

In the table 7.2, we can see the average and maximal cardinality of a partition, Z_i and the sets which are used to compute Z_i. The symbol $\varnothing$ corresponds to an average value, $\widetilde{t}$ is a median and max refers to a maximal measured value.

To understand this influence more deeply, we can observe how the cardinality of Z_i depends on the number of iteration in Figure 7.4.

We have observed that at first, the cardinality of $\lfloor Z_i \rfloor$ and $\lceil Z_i \rceil$ is prone to increase, which signifies the dramatic rise of the cost of an intersection and complementation. However, in case of abstract bidirectional algorithms, both computed sets $Z_i^{\rightarrow}$ and $Z_i^{\leftarrow}$ do not grow so fast, which explains the better results in context of them.

Thus, we consider few practical examples of evaluating the emptiness test of an *AFA* to decide whether the impact of costly intersection and complementation computation would be acceptable in comparison with operations which are performed by concrere algorithms.

	A. Forw.			A. Back.			Bi. Forw.			Bi. Back.		
	$\varnothing$	$\widetilde{t}$	max	$\varnothing$	$\widetilde{t}$	max	$\varnothing$	$\widetilde{t}$	max	$\varnothing$	$\widetilde{t}$	max
i	2.4	2	20	2.3	2	13	1.4	1	7	1.4	1	7
$\|P_i\|$	7.5	5	30	5.4	4	29	7.7	6	30	7.6	6	30
$\|\lfloor Z_i \rfloor\|$	72	43	856	-	-	-	29.6	27	277	-	-	-
$\|\lceil Z_i \rceil\|$	-	-	-	10.4	2	1398	-	-	-	2	1	52
$\|\lfloor \gamma_{P_i}(.) \rfloor\|$	56.4	35	627	-	-	-	33.5	27	239	-	-	-
$\|\lceil \gamma_{P_i}(.) \rceil\|$	-	-	-	6.3	1	547	-	-	-	2.5	1	302
$\|\lfloor \widetilde{pre}_A(.) \rfloor\|$	76.5	45	856	-	-	-	42.4	32	256	-	-	-
$\|\lceil \widetilde{post}_A(.) \rceil\|$	-	-	-	27	4	1698	-	-	-	16.1	2	1119

Table 7.2: Subresults computed within an execution of studied algorithms

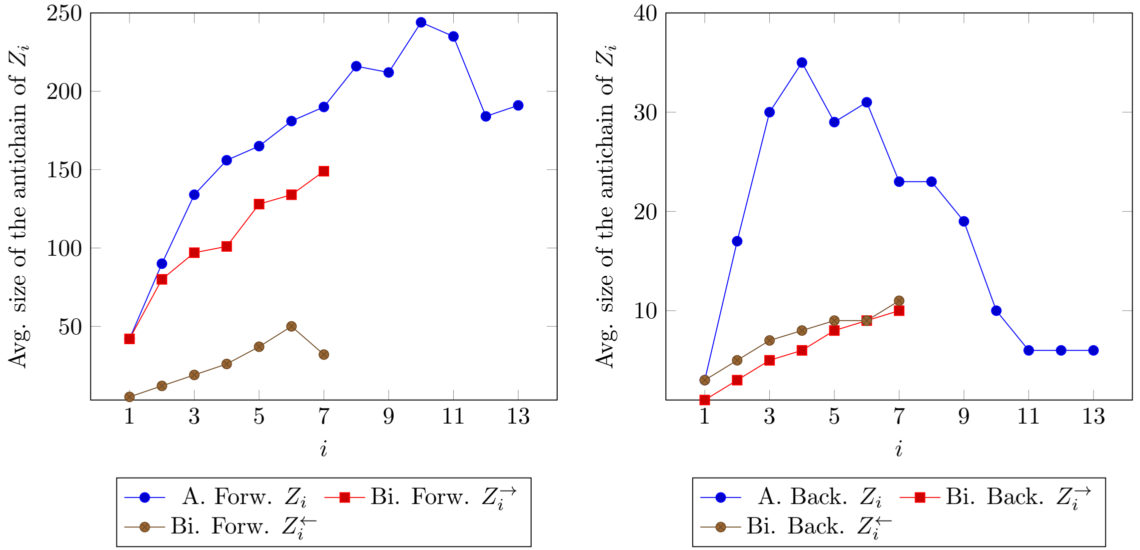

Figure 7.4: Average cardinality of a set $\lfloor Z_i \rfloor$ or $\lceil Z_i \rceil$ with respect to a iteration i of abstract algorithms

7.4 Comparison of Studied Algorithms in Context of Emptiness of NFA Intersection

.

Definition 51 *Let A, B be two AFAs. In what follows, the **pseudointersection** of two* AFA*s A, B, will be every* AFA $A \widetilde{\cap} B$, *which satisfy the property that $L(A \widetilde{\cap} B) = \varnothing \Leftrightarrow$ $L(A) \cap L(B) = \varnothing$. In case of $L(A \widetilde{\cap} B) \neq \varnothing$, the language of $L(A \widetilde{\cap} B)$ could be equal to anything except the empty language.*

Suppose that $A = (Q_A, \Sigma_A, S_A, R_A, F_A)$ and $B = (Q_B, \Sigma_B, S_B, R_B, F_B)$ are two *NFA*s, where $Q_A \cap Q_B = \varnothing$, $a \in \Sigma_A$ and $s_0, a_0, b_0 \notin Q_A \cup Q_B$. Then, we can replace the image of the transition relations with logical variables conncected by conjunctions and obtain two *AFA*s $A^{AFA} = (Q_A, \Sigma_A, S_A, \delta_A, F_A)$, $B^{AFA} = (Q_B, \Sigma_B, S_B, \delta_B, F_B)$, where $L(A) = L(A^{AFA})$ and

$L(B) = L(B^{AFA})$. To decide whether $L(A) \cap L(B) = \varnothing$, it is sufficient to find out whether $L(A^{AFA} \,\tilde{\cap}\, B^{AFA}) = \varnothing$. Thus, we present an algorithmic way to create a pseudointersection of given AFAs without evaluating their language.

To achieve this goal, we can simply consider three new transitions $U = \{(s_0, a, \phi), (a_0, a, \phi_a), (b_0, a, \phi_b)\}$, where $\phi = a_0 \wedge b_0, \phi_a = \bigvee_{s \in S_A} s$ and $\phi_b = \bigvee_{s \in S_B} s$ and create an automaton $A^{AFA} \,\tilde{\cap}\, B^{AFA} = (Q_A \cup Q_B \cup \{s_0, a_0, b_0\}, \Sigma_a \cup \Sigma_b, \{s_0\}, \delta_A \cup \delta_b \cup U, F_A \cup F_B)$.

Notice that we can use this method to create a pseudointersection of more than 2 NFAs in once. In what follows, we will call each former NFA which was used to create a pseudointersection a *component* of an intersection.

Suppose that $T \subseteq \mathbb{ETV}$ are all the generators of extended Tabakov-Vardi model such that $s = 10, r \in \{5, 10, 15, 20\}, f = 0.1, c = 0.0, p \in \{0, 0.5, 1\}$. Then, we generate 60 AFAs using each generator $G \in T$ and we separate them into 20 triplets (A, B, C). It is easy to see that we are able to construct 240 AFAs $A \,\tilde{\cap}\, B \,\tilde{\cap}\, C$. Let us execute studied and proposed algorithms over these pseudointersections of AFAs.

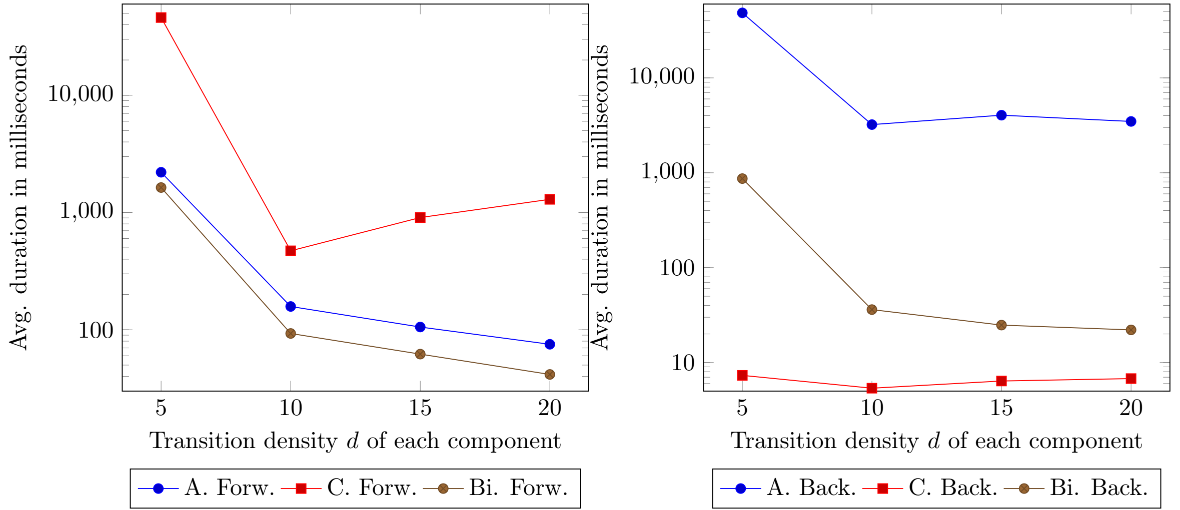

Figure 7.5: Average execution time of an emptiness test of 3 NFAs intersection over 240 triplets of random NFAs

The results are summarized in Figure 7.5, which shows the dependency between transition density of each component of an pseudointersection and the average time of evaluation the emptiness tests. We have observed that in case of the forward fashion, the abstract algorithms gave us significantly better results than the concrete one. We can conclude that the growth of the transition density of each component of an pseudointersection even reduces the cost of abstract forward and abstract bidirectional forward-like tests.

However, the emptiness tests working in backward fashion brought us completely different results. We have found out that the concrete backward test still works much more effective than the abstract backward ones. Nevertheless, we can observe again that the cost of these algorithms decreases when the transition density of each component grows.

7.5 Comparison of Studied Algorithms in Context of NFA Inclusion

Let A, B be two *NFA*s. To determine whether $L(A) \subseteq L(B)$, we can use the formula

$$L(A) \subseteq L(B) \iff L(A) \cap \overline{L(B)} = \varnothing.$$

We can again reuse the idea of simple *NFA* to *AFA* conversion and pseudointersection presented in 7.4. Then, we conclude that

$$L(A) \subseteq L(B) \iff L(A \mathbin{\tilde{\cap}} \overline{B}) = \varnothing.$$

Let us discuss the possibility of converting an *AFA* $B = (Q, \Sigma, S_0, \delta, F)$, where $sink \notin Q$ to its complement $\overline{B}$, such that $L(B) = \overline{L(\overline{B})}$. First, we convert B to a corresponding *AFA* $B' = (Q \cup \{sink\}, \Sigma, S_0, \delta', F)$, such that we state $\delta' = \{(q, a, \phi) \in \delta \mid \phi \neq \bot\} \cup \{(q, a, sink) \mid \exists (q, a, \phi) \in \delta : \phi = \bot\} \cup \{(sink, a, sink) \mid a \in \Sigma\}$.

Then, we create a function δ'' by transforming each element $(q, a, \phi) \in \delta'$. We replace each occurrence of $\wedge$ by $\vee$ in ϕ and vice versa. Since we use only formulae in DNF in our implementation, we obtain a CNF by this procedure, which will be then converted back to DNF using the distributive laws of Boolean algebra [5].

Finally, we can state that $\overline{B} = (Q \cup \{sink\}, \Sigma, S_0, \delta'', Q \setminus F)$.

In what follows, we consider $T \subseteq \mathbb{ETV}$ to be set of all generators of extended Tabakov-Vardi model, such that $s \in \{10, 15, 20\}, d \in \{1, 2, 5, 10, 20\}, p \in \{0, 0.5, 1\}, c = 0.0, f = 0.1$. For each $G \in T$, we generate 40 *AFA*s and we separate them to 20 triplets (A, B). We are thus able to construct 900 *AFA*s $A \mathbin{\tilde{\cap}} \overline{B}$. Let us execute studied and proposed algorithms over these pseudointersections of *AFA*s.

The measured results are depicted in the following charts. In Figure 7.6, we show how the average duration of studied algorithms evaluation depends on a state-set cardinality of each compoment of a pseudointersection. We have observed that the abstract algorithms do not reduce the cost of emptiness test in context of deciding *NFA* inclusion, if a state-set cardinality is our main criterion.

Thus, we tried to summarize the dependency between a transition density and an average duration of studied algorithms execution. The graphs shown in Figure 7.7 expresses this dependency. We can conclude that in context of concrete algorithms, the average duration of concrete algorithms execution increases while the transition density grows, while the average duration of abstract algorithms execution tends to decrease. In case of $d = 20$, the abstract bidirectional forward-like algorithm brougth us better results than the concrete one. However, the concrete backward algorithm remains unbeaten.

Next, we have chosen a single generator $G \in \mathbb{ETV}$, where $s = 30, f = 0.1, p = 0.5, c = 0.0$ and $d = 10$ to generate 800 tuples (A, B) of *AFA* to test our hypothesis about the influence of a high transition density to average duration of our algorithms.

To achieve this goal, we have used our optimizations presented in Section 7.3. Concretely, we have chosen the second optimization of the abstract forward algorithm and the second

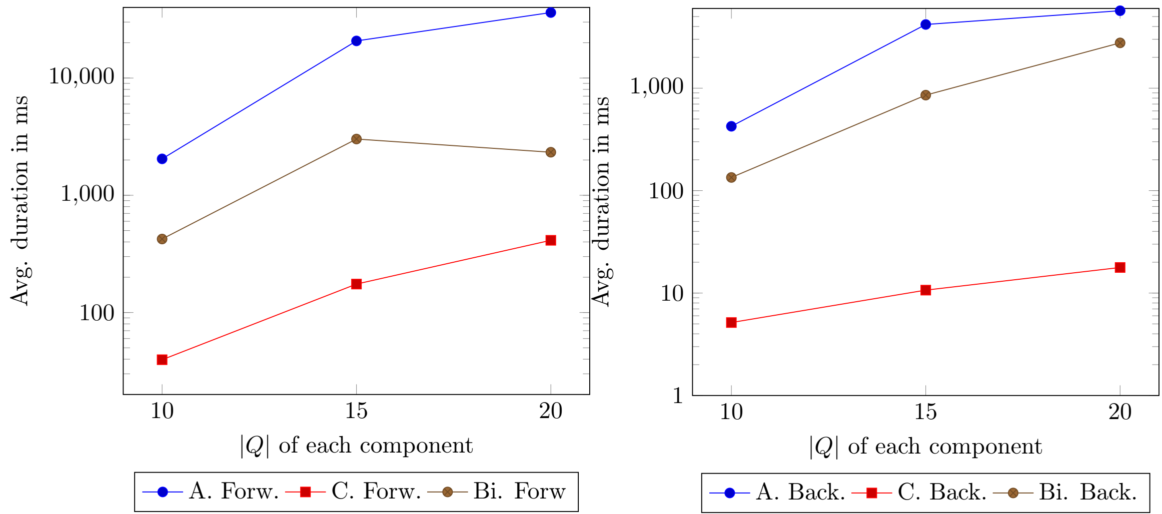

Figure 7.6: Average execution time of *NFA* inclusion test over 900 tuples of random *AFA*s

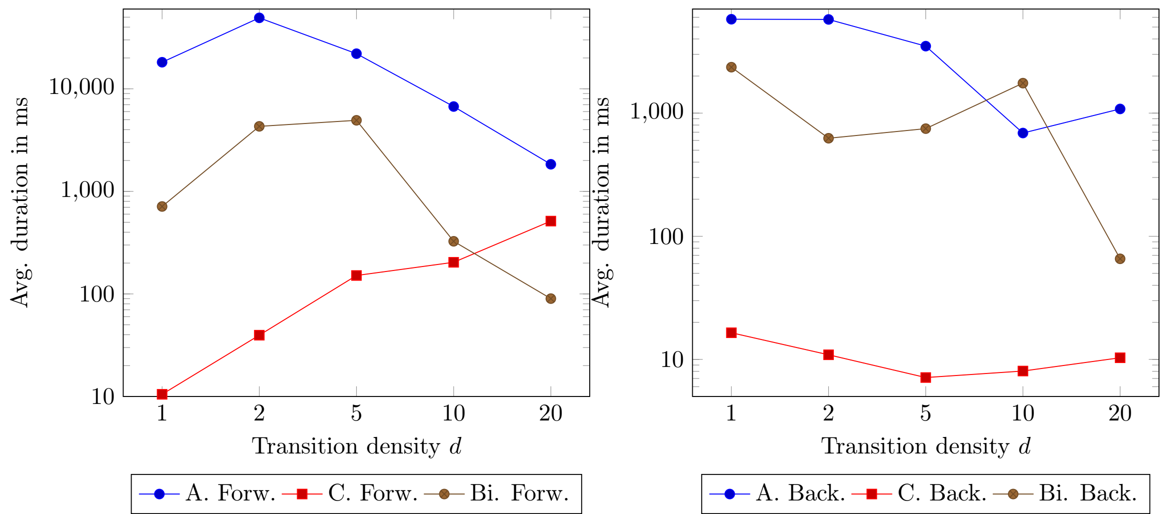

Figure 7.7: Average execution time of *NFA* inclusion test over 900 tuples of random *AFA*s

optimization of the bidirectional abstract forward-like algorithm to test a language inclusion between *NFA*s with 30 states.

In Figure 7.8 on left, we can see the comparison between second optimization of the abstract forward algorithm and the concrete forward algorithm. Note that the scale of the axis is logarithmic. We have observed that in case of a huge transition density $d = 10$, there is a significant difference between execution of a concrete forward test over automata which are empty and over automata which are non-empty. In case of non-empty automata, the concrete forward test terminates very fast, while in case of empty automata, the abstract algorithms gives us better results.

Finally, we can notice in Figure 7.9, that the abstract bidirectional forward-like algorithm gives us slightly better results in case of non-empty automata than the former abstract one.

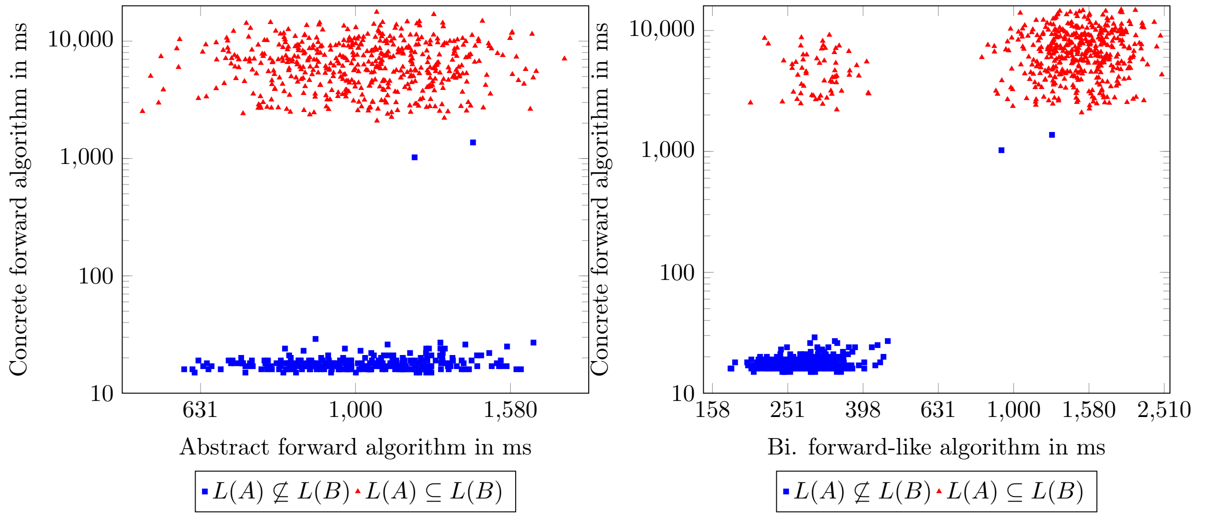

Figure 7.8: Comparison of forward algorithms over 800 instances of *NFA* inclusion problem

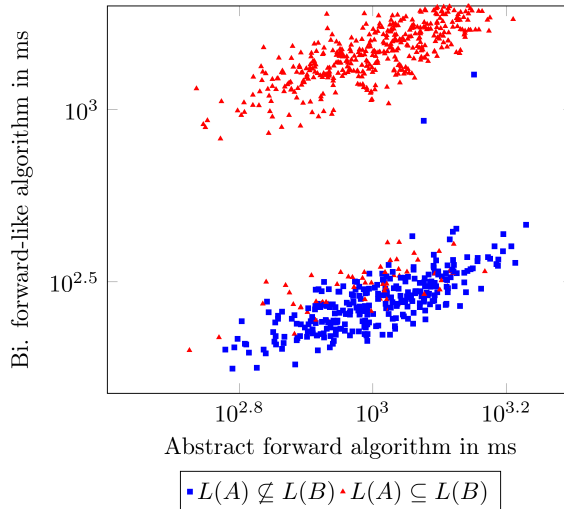

Figure 7.9: Comparison of forward algorithms over 800 instances of *NFA* inclusion problem

Chapter 8

Conclusion

In this thesis, we have presented several algorithms for testing emptiness of alternating finite automata and we experimentally evaluated the performance of these studied algorithms over thousands of randomly generated automata.

First, we have introduced the crucial theoretical background about *NFA*s and *AFA*s and we discussed the necessity not to use the naive *AFA* to *NFA* conversion to decide an *AFA* emptiness. Thus, we presented so-called antichain-based *concrete forward* and *concrete backward* algorithms which avoid the explicit *AFA* to *NFA* conversion.

Next, we presented both *abstract forward* and *abstract backward* algorithms which come up with the idea of a abstract domain of states to reduce a size of given *AFA*s and possibly facilitate the emptiness test. We have modified the studied algorithms to make them work properly, since the former version gave us wrong results in some edge cases.

Subsequently, the author of this thesis proposed his own *abstract bidirectional forward-like* and *abstract bidirectional backward-like* algorithms for deciding *AFA* emptiness, which are based on the concept of the concrete ones. The proposed algorithms take advantage of the former abstract algorithms and compute additional useful information which can possibly accelerate the process of the *AFA* emptiness testing.

Then, we have discussed the efficient way to represent all the data structures within the implementation of presented algorithms. The author proposed his own way to efficiently compute an inverse transition relation of an *AFA* to enable the possibility to effectively inspect the behaviour of an *AFA*.

Finally, we have introduced a Tabakov-Vardi model for generating random *NFA*s and we proposed its extension to be able to generate *AFA*s and influence the process of random generating by more parameters. We have used this extended model to generate thousands of random *AFA*s and we have experimentally evaluated studied algorithms over these *AFA*s.

We have observed that in case of total random *AFA*s, the abstract algorithms do not bring us any advantage in context of evaluation speed and that the former concrete algorithms work more efficiently.

Thus, we have inspected the behaviour of abstract algorithms in detail to find out which operations correspond to the greatest efficiency bottleneck. It was discovered that the computation of closed set intersection and closed set complementation massively decelerate

the whole procedure. To deal with this problem, we have figured out few optimizations of both former and proposed abstract algorithms to avoid computing an intersection of closed set so often. One of the optimizations brings the idea of giving up on precision of iteratively computed information in exchange for the possibility not to perform costly closed set intersection over large sets. We have observed that this optimization significantly reduce the cost of presented abstract algorithms and in case of huge transition density, it is possibly able to work faster than the concrete algorithms.

Next, we considered several real-world problems which are connected to the *AFA* emptiness test. Concretely, we have introduced the *NFA* inclusion and the *NFA* intersection problems which are solvable using studied algorithms. We have generated hundreds of *NFA*s to experiment with. We have the observed that in context of intersection of several *NFA*s, the forward abstract algorithms give us significantly better results than the concrete forward one and they tend to improve when increasing transition density. However, the concrete backward algorithm still works better than the abstract backward ones.

In future, we could try to find more classes of *AFA*s which reduce the cost of the abstract algorithms in comparison to the concrete ones. We could also design another model for generating random *AFA*s to somehow disadvantage the concrete backward algorithm or even evaluate studied algorithms over real-world benchmarks. Next, we can figure out an improvement for the abstract backward algorithms to avoid double evaluating of a complement of a closed set when computing a new abstract domain.

Bibliography

[1] Burris, S. and Sankappanavar, H. P. *A Course in Universal Algebra.* Springer, 1981. Available at: http://www.math.uwaterloo.ca/~snburris/htdocs/ualg.html.

[2] Erné, M., Koslowski, J., Melton, A. and Strecker, G. E. A primer on Galois connections. *Annals of the New York Academy of Sciences.* Blackwell Publishing Ltd Oxford, UK. 1993, vol. 704, no. 1, p. 103–125.

[3] Fisher, C., Fogarty, S. and Vardi, M. Random Models for Evaluating Efficient Büchi Universality Checking. In: Ghosh, S. and Prasad, S., ed. *Logic and Its Applications.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, p. 91–105. ISBN 978-3-662-54069-5.

[4] Ganty, P., Maquet, N. and Raskin, J. Fixed point guided abstraction refinement for alternating automata. *Theoretical Computer Science.* 2010, vol. 411, p. 3444–3459. DOI: https://doi.org/10.1016/j.tcs.2010.05.037.

[5] Halmos, P. and Givant, S. *Introduction to Boolean Algebras.* Springer New York, 2009. 8–13 p. Available at: https://doi.org/10.1007/978-0-387-68436-9.

[6] Larsen, K. S. *A Note on Lattices and Fixed Points.* 2007.

[7] Meduna, A. *Automata and languages : theory and applications.* London New York: Springer, 2000. ISBN 978-1852330743.

[8] Tabakov, D. and Vardi, M. Y. Experimental Evaluation of Classical Automata Constructions. In: Sutcliffe, G. and Voronkov, A., ed. *Logic for Programming, Artificial Intelligence, and Reasoning.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, p. 396–411. ISBN 978-3-540-31650-3.

[9] Tarski, A. et al. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics.* Pacific Journal of Mathematics. 1955, vol. 5, no. 2, p. 285–309.