Bachelor's Thesis Computing Science

Radboud University Nijmegen

Regular Languages and Finite Monoids

Author:
Lucas Maarse
s1101952

First supervisor/assessor:
dr. Jana Wagemaker

Second assessor:
dr. Jurriaan Rot

November 23, 2025

Abstract

The theorem of equivalence of finite monoids and regular languages is well known, and has been very useful for efficient algorithms such as algorithms on strings and trees and an algorithm for the equational theory of binary relations. In this thesis, we give an elaborate proof for this theorem. Additionally, we give also give an explained proof for the equivalence of context-free grammars and groupoids, more commonly known as magmas, using grammars. We also create an alternate proof for part of the first theorem, by creating a regular grammar from a finite monoid.

Contents

Chapter 1

Introduction

The Chomsky hierarchy is a hierarchy of classes of formal grammars in order of complexity. In the Chomsky hierarchy, the most simple or restricted type of languages are regular languages. They are used in many different areas, such as lexical analysis in programming language compilation, circuit design [3], and pattern matching [8].

The most known ways to define and recognize regular languages are regular expressions and finite automata. The equivalence of regular expressions and finite automata is known as Kleene's Theorem [7]. Another known way to recognize regular languages is by using a finite monoid and a homomorphism [4].

Using monoids to recognize regular languages can be very useful. Algebraic representations for regular languages such as monoids can be used for efficient algorithms such as can be seen by several algorithms on strings and trees implemented using monoids [2] and an algorithm for the equational theory of binary relations which is of PSpace complexity. [11].

There are many theorems that followed from the theorem about finite monoids recognizing regular languages. For example, it was generalized to a theorem about context-free languages and finite groupoids [1]. Additionally, there exists a theorem about piecewise-testable languages and J-trivial monoids [13]. Another example is a theorem about star-free languages and aperiodic monoids, which are subsets of regular languages and monoids [12].

The class of languages just above regular languages in the Chomsky hierarchy are the context-free languages. Similar to using finite monoids to recognize regular languages, we can use finite groupoids to recognize context-free languages. Recognizing context-free languages using finite groupoids was used to propose extensions to Barrington's M-model, a model of computa-

tion. An M-program loosly consists of sequences of simple instructions, one for each input length, which translates an input string into a word over a monoid, and then evaluates it in the monoid which determines acceptance of the input. The extensions consider using groupoids instead of monoids [1].

In this thesis, I provide a well-explained proof for the theorem of monoids recognizing regular languages. We see that from a finite automaton, we can create a finite monoid that recognizes the same language, and likewise that we can create a finite automaton from a finite monoid. This proves that finite monoids recognize exactly regular languages. This part mostly follows the short proof outline of [11].

Additionally, I provide a well-explained proof for the theorem of finite groupoids recognizing context-free languages. We see that from a finite groupoid, we can create a context-free grammar, and also the other way around. This proves that finite groupoids recognize context-free languages. This part follows the short proof outline of [1].

Finally, I also give a proof for the statement that when a language is recognized by a finite monoid, it is also generated by a regular grammar. This proof is based on the proof for finite groupoids and context-free languages, but is adapted for finite monoids.

Chapter 2

Preliminaries

In this chapter, we note a few definitions necessary to understand the other chapters, and a few examples to understand them better.

First, we define a non-deterministic automaton.

Definition 2.1. A non-deterministic automaton is a tuple with the following elements:

non-empty set S	the set of states of the automaton
non-empty set Σ	the alphabet of the automaton
two subsets $I, F \subseteq S$	I is the set of initial states and F is the set of final states
function δ: $S \times \Sigma \to \mathcal{P}(S)$	the set of transitions of the automaton

The power set $\mathcal{P}(S)$ is the set that contains all subsets of the set S. The automaton is finite if and only if the set of states S is finite. Alternatively, we may define a deterministic automaton, where δ is of type $\delta \subseteq S \times \Sigma \times S$ and I is one element of S instead [6] [4]. We may abbreviate non-deterministic finite automata and deterministic finite automata as NFA and DFA respectively.

We now define words.

Definition 2.2. A word of a finite set Σ is any sequence of zero or more elements of Σ. We denote the set of words over Σ as the set Σ^*. The word with zero letters is called the empty word and may be denoted with ε.

We may also define the length of a word.

Definition 2.3. The length of a word $w \in \Sigma^*$, also written as $|w|$, is equal to the amount of letters in Σ this word contains.

We define a new recursive function using δ, for both deterministic and non-deterministic automata.

Definition 2.4. Assume an NFA $(S, \Sigma, I, F, \delta)$. The transition function δ^*: $S \times \Sigma^* \to \mathcal{P}(S)$ takes a word w and state s, and maps these to a set with all states reachable from s by taking a w-labeled path. It is a recursive function defined as follows.
$\forall w \in \Sigma^*$, $\forall a \in \Sigma$ and $\forall s \in S$

- $\delta^*(\varepsilon, s) = \{s\}$
- $\delta^*(a, s) = \delta(a, s)$
- $\delta^*(aw, s) = \{q | q \in \delta^*(w, s') \wedge s' \in \delta(a, s)\}$

We can also define the recursive function for a deterministic finite automaton.

Definition 2.5. Assume an DFA $(S, \Sigma, I, F, \delta)$. The transition function δ^*: $S \times \Sigma^* \to S$ takes a word w and state s, and maps these to the state reachable from s by taking a w-labeled path. It is a recursive function defined as follows.
$\forall w \in \Sigma^*$, $\forall a \in \Sigma$ and $\forall s \in S$

- $\delta^*(\varepsilon, s) = s$
- $\delta^*(a, s) = \delta(a, s)$
- $\delta^*(aw, s) = \delta^*(w, \delta(a, s))$

An automaton recognizes a word $w \in \Sigma^*$ if there exists a w-labeled path from an initial state to a final state. We formally define the language recognized by an automaton.

Definition 2.6. The language recognized by an NFA $(S, \Sigma, I, F, \delta)$ is the set $L = \{w | f \in \delta^*(w, i) \wedge f \in F \wedge i \in I\}$.

Definition 2.7. The language recognized by an DFA $(S, \Sigma, I, F, \delta)$ is the set $L = \{w | f = \delta^*(w, I) \wedge f \in F\}$.

Now, we introduce monoids.

Definition 2.8. A monoid is a tuple with three elements, (M,$\cdot$, and 1), where M is a set, and $\cdot$ is an associative binary operation on M with 1 as neutral/identity element. The following needs to hold $\forall m, n, p \in M$.

- $(m \cdot n) \cdot p = m \cdot (n \cdot p)$
- $m \cdot 1 = 1 \cdot m = m$

Sometimes, "$\cdot$" may be omitted which means that $n \cdot m$ may be written as nm instead. The neutral element may be denoted as 1_M to avoid ambiguity when necessary. The monoid is finite if M is finite [6].

In order to better understand monoids, we look at a few examples.

Example 2.8.1. Take the monoid with $M = \{true, false\}$ with binary operator OR. The following defines the operator OR:

$$\begin{aligned} true \text{ or } true &= true \\ false \text{ or } true &= true \\ true \text{ or } false &= true \\ false \text{ or } false &= false \end{aligned}$$

From this definition, it follows that the neutral element is $false$. It is clear the OR operation is associative. Since M is finite, the monoid is finite.

Example 2.8.2. Take the monoid with $M = \mathbb{Z}$, the set of integers, with binary operator multiplication. The neutral element is 1 as $x \cdot 1 = 1 \cdot x = x$. The operation is clearly associative. This monoid is not finite.

Example 2.8.3. Take the monoid with $M = \{0, 1, 2, 3\}$ with binary operator addition modulo four: $a \cdot b = (a + b) \bmod 4$. The neutral element is 0 as $x \cdot 0 = 0 \cdot x = x$. We can see that the operator is associative from how modulo works. This monoid is finite.

We now define a special and very useful type of monoid, the free monoid.

Definition 2.9. The free monoid on a set A is the monoid whose elements are all finite sequences of elements from A, with concatenation as the binary operator. The sequence of zero elements is the identity element, often denoted as ϵ. The free monoid on A is usually denoted as A^*. Elements of a free monoid may be known as words [9].

We now define a special type of map, the monoid homomorphism.

Definition 2.10. Let M and N be two monoids. Then the map $m : M \to N$ is a monoid homomorphism if it maps the neutral element of M to the neutral element of N and if it respects multiplication [6]. Thus the following needs to hold for a map to be a monoid homomorphism:

- $\forall p, q \in M, (m(p \cdot q) = m(p) \cdot m(q))$
- $m(1_M) = 1_N$

We look at a few examples of homomorphisms.

Example 2.10.1. We take the monoid for addition of natural numbers $(\mathbb{N}, \cdot, 0)$, and the monoid for addition of natural numbers modulo 4, $(M, \cdot, 0)$ with $M = \{0, 1, 2, 3\}$. We can define a homomorphism h: $\mathbb{N} \to M$ as follows:

$$h(x) = x \bmod 4$$

Clearly, the neutral element of $\mathbb{N}$ is mapped to that of M, since $h(0) = 0$.

Additionally, it does not matter whether we first apply h or not, since $(x+y)$ mod $4 = (x \bmod 4) + (y \bmod 4) \bmod 4$. Therefore, h indeed is a homomorphism.

We define the preimage of a homomorpism. We denote it similarly to the inverse of a map. However, it works somewhat differently than the inverse.

Definition 2.11. Take a homomorphism $h : A \to B$. We then define the preimage $h^{-1} : B \to \mathcal{P}(A)$ as $h^{-1}(p) = \{w | h(w) = p\}$

Next we define the recognition of languages using monoids.

Definition 2.12. A monoid M recognizes a set $L \subseteq \Sigma^*$ if there is a homomorphism h and a subset $P \subseteq M$ such that $L = h^{-1}(P)$ [4].

Example 2.12.1. Let us take $\Sigma = \{a, b\}$, $L = \{a, aa, b\}$ and a monoid $M = \{1, 2, 3, 4\}$ with a homomorphism $h : \Sigma^* \to M$. Let us take $P = \{1, 2\} \subseteq M$. If we have for example $h(a) = 1, h(aa) = h(b) = 2$, and for any other word $w \in \Sigma^*$ that $h(w) \in \{3, 4\}$, the monoid recognizes the language since $L = h^{-1}(P)$.

It is useful to recall the definition of a regular language.

Definition 2.13. A language $L \subseteq \Sigma^*$ is regular if and only if it is recognized by a finite automaton.

This definition is equivalent to saying a language is regular if and only if it is generated by a regular expression, and also if and only if it is generated by a regular grammar.

As as we prove later in the thesis, a language is also regular if it is recognized by a finite monoid.

Chapter 3

Regular languages and finite monoids

In this chapter, we provide a proof of the theorem that monoids recognize regular languages, and explain this proof. In this chapter, we let L be a language over a finite alphabet Σ. Formally, we can state the theorem as follows.

Theorem 3.1. *Let L be a language over a finite alphabet Σ. The language L is regular if and only if L is recognized by a finite monoid M as defined in 2.12.*

The theorem can be split into two implications. We prove both of these implications in two sections. We use finite automata to prove these implications, since we know finite automata recognize regular languages.
In the first section we take an NFA and create a finite monoid from it. We then show that the monoid indeed recognizes the same language as the NFA.

In the second section we do the opposite, taking a finite monoid and creating an DFA from it, and showing the DFA recognizes the same language as the monoid.

3.1 From finite automaton to finite monoid

In the first part, we provide an explanation for the proof of first implication of the theorem. We can describe this as follows.

Theorem 3.1.1. *Let L be a language over a finite alphabet Σ. If the language L is regular then L is recognized by a finite monoid.*

We will prove this by showing we can create a finite monoid from any finite automaton, recognizing the same language.

Defining the monoid

We will define an arbitrary NFA and a monoid from this NFA. First, we take an arbitrary NFA $(S, \Sigma, I, F, \delta)$. Using this NFA, we define a monoid $(M, \cdot, 1)$.

Definition 3.1.2 (monoid of an automaton)**.** We define the monoid $(M, \cdot, 1)$ of an automaton $(S, \Sigma, I, F, \delta)$ as follows:

- $M = \mathcal{P}(S \times S)$
- $\forall R, S \in M, R \cdot S = \{(r_1, s_2) | (r_1, r_2) \in R \wedge (s_1, s_2) \in S \wedge r_2 = s_1\}$
- $1 = \{(s, s) | s \in S\}$

We now prove that this indeed forms a monoid.

Lemma 3.1.3. *The monoid $(M, \cdot, 1)$ of an automaton $(S, \Sigma, I, F, \delta)$ is indeed a monoid.*

Proof. For M to be a monoid, associativity of the binary operator and identity of the identity element of the operator have to be proven. We provide the two proofs below.

Associativity property
$\forall m, n, p \in M, (m \cdot n) \cdot p = m \cdot (n \cdot p)$

The following follows from the definition of the operator:

$$\begin{aligned} m \cdot n &= \{(m_1, n_2) | (m_1, m_2) \in m \wedge (n_1, n_2) \in n \wedge m_2 = n_1\} \\ n \cdot p &= \{(n_1, p_2) | (n_1, n_2) \in n \wedge (p_1, p_2) \in p \wedge n_2 = p_1\} \end{aligned}$$

By combining definitions, we can rewrite $(m \cdot n) \cdot p$.

$$\begin{aligned} (m \cdot n) \cdot p &= \{(m_1, p_2) | (m_1, n_2) \in m \cdot n \wedge (p_1, p_2) \in p \wedge n_2 = p_1\} \\ &= \{(m_1, p_2) | (m_1, m_2) \in m \wedge (n_1, n_2) \in n \wedge m_2 = n_1 \\ &\quad \wedge (p_1, p_2) \in p \wedge n_2 = p_1\} \end{aligned}$$

Similarly, we can also rewrite $m \cdot (n \cdot p)$.

$$\begin{aligned} m \cdot (n \cdot p) =& \{(m_1, p_2) | (m_1, m_2) \in m \wedge (n_1, p_2) \in n \cdot p \wedge m_2 = n_1\} \\ =& \{(m_1, p_2) | (m_1, m_2) \in m \wedge (n_1, n_2) \in n \wedge (p_1, p_2) \in p \\ & \wedge n_2 = p_1 \wedge m_2 = n_1\} \end{aligned}$$

It becomes evident that the property holds, since both sides can be rewritten to an equivalent statement.

Identity property
$m \cdot 1 = 1 \cdot m = m$

We can prove this property by rewriting using the definition of 1.

$$\begin{aligned} m \cdot 1 &= \{(m_1, s_2) | (m_1, m_2) \in m \wedge (s_1, s_2) \in 1 \wedge s_1 = m_2\} \\ &= \{(m_1, m_2) | (m_1, m_2) \in m \wedge (m_2, m_2) \in 1\} \\ &= \{(m_1, m_2) | (m_1, m_2) \in m\} \qquad (*) \\ &= m \end{aligned}$$

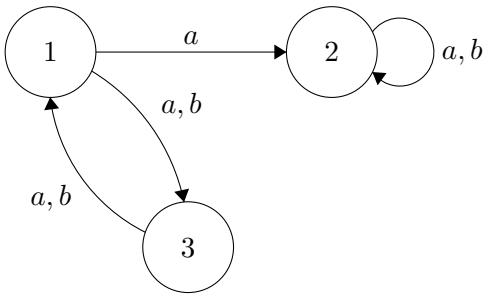
At (*) we can rewrite since $(m_2, m_2) \in 1$ is always true from the definition of 1.

Similarly, it can also be proven that $1 \cdot m = m$. Therefore, 1 is indeed the identity element of our monoid M. □

In addition to M being a monoid, it is also clear it is finite. The powerset and cartesian products of finite sets form another finite set. Therefore $M = \mathcal{P}(S \times S)$ is finite.

Homomorphism

For a monoid to recognize a language, we need a homomorphism that can map elements from the language to elements from the monoid. We show there exists such a homomorphism. First, we will take a look at an example of an automaton showing how the homomorphism works, and later we formally define it.



See the automaton above. If we take the word b, we are able to go from state 1 to state 3, state 3 to state 1, or from state 2 to state 2. If we take the word a we can do the same, but we are also able to go from state 1 to state 2.

Seen these transitions, we could map the word b to $\{(1,3),(2,2),(3,1)\} \in M$. We can do the same for a and words consisting of multiple letters, such as $\{(1,1),(2,2),(3,3)\}$ for the word bb. In the next section, we formally define the map which maps words to an element of the monoid.

Defining the homomorphism

We first recall the alphabet Σ of our NFA. Then Σ^* forms the free monoid over Σ with all words over Σ. We can now define a map $h : \Sigma^* \to M$, that will map each word to its transition relation:

Definition 3.1.4 (map for a monoid of an automaton)**.** *Take an automaton $(S, \Sigma, I, F, \delta)$ and its monoid $(M, \cdot, 1)$. We then define the map for the monoid of an automaton $h : \Sigma^* \to M$ as the map $h(w) = \{(s,q) | q \in \delta^*(w,s)\}$.*

As stated earlier, to let a monoid recognize a language, we need this map to be a homomorphism. Therefore, we will prove the following lemma.

Lemma 3.1.5. *the map h for a monoid $(M, \cdot, 1)$ of an automaton $(S, \Sigma, I, F, \delta)$ as defined in 3.1.6 is indeed a homomorphism.*

Proof. To prove the map h is indeed a homomorphism, we need to prove two properties.

The first property is $h(1_{\Sigma^*}) = 1_M$ and can easily be proven by applying the homomorphism h on $1_{\Sigma^*} = \varepsilon$:

$$\begin{aligned} h(\varepsilon) &= \{(s,q) | q \in \delta^*(\varepsilon, s)\} \\ &= \{(s,q) | q \in \{s\} \wedge s \in S\} \\ &= \{(s,s) | s \in S\} \\ &= 1_M \end{aligned}$$

The second property that should hold is $\forall v, w \in \Sigma^*, h(v \cdot w) = h(v) \cdot h(w)$.

We prove this by induction. The base case is $v = \varepsilon$. The induction step will show that the property holds for words (av) and w where $a \in \Sigma$. For this, the induction hypothesis used is that the property holds for words v and w.

Base case:
We have $v = \varepsilon$ and any $w \in \Sigma^*$. Considering this is the identity element, we can prove the base case as follows.

$$\begin{aligned} h(\varepsilon w) &= h(w) \\ &= 1_M \cdot h(w) \\ &= h(\varepsilon) \cdot h(w) \end{aligned}$$

Induction step:
For the induction step let $a \in \Sigma$ and $v, w \in \Sigma^*$.

Our induction hypothesis (IH) is $h(vw) = h(v) \cdot h(w)$.

Before proving that $h(avw) = h(av) \cdot h(w)$, we first prove that $h(av) = h(a) \cdot h(v)$ for any word v.

$$\begin{aligned} h(av) &= \{(s,q) | q \in \delta^*(av, s)\} \\ &= \{(s,q) | q \in \delta^*(v, s') \wedge s' \in \delta(a, s)\} \\ &= \{(s,q) | s' \in \delta^*(a, s) \wedge q \in \delta^*(v, s')\} && (*) \\ &= \{(s,q) | s' \in \delta^*(a, s) \wedge (s', q) \in h(v)\} \\ &= \{(s,q) | (s, s') \in h(a) \wedge (s', q) \in h(v)\} \\ &= \{(s,q) | (s, a_2) \in h(a) \wedge (v_1, q) \in h(v) \wedge a_2 = v_1\} \\ &= h(a) \cdot h(v) \end{aligned}$$

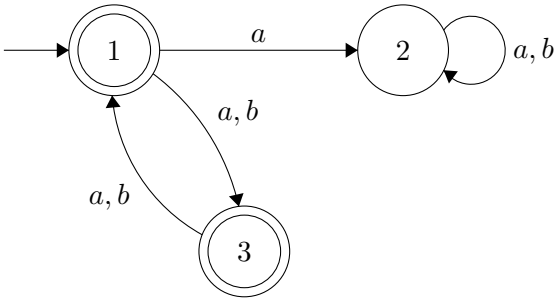
(*): From the definitions we know that $\delta^*(a, s) = \delta(a, s)$, when $a \in \Sigma$.
We can now finish the induction step using the induction hypothesis.

$$\begin{aligned} h((av)w) &= h(a(vw)) \\ &= h(a) \cdot h(vw) \\ &= h(a) \cdot h(v) \cdot h(w) && \text{(IH)} \\ &= h(av) \cdot h(w) \end{aligned}$$

This concludes the induction proof. Since both properties have been proven, the map h is indeed a homomorphism. □

Recognizing the language using the monoid

Recall the definition for a monoid recognizing a language; for a monoid M to recognize a language L, there needs to be a set $P \subseteq M$ and a homomorphism h so that $L = h^{-1}(P)$. First we give an example of an automaton and a set P, and afterward we define this set formally.



See the automaton above. It is similar to the previous automata, but the initial states and the final states are indicated. The initial state is 1 and the final states are 1 and 3. We define a set P, containing all relations with (1,3) or (1,1) in it.

Words such as b and bab are mapped to a relation with (1,3). Words such as ε and bb are mapped by the homomorphism of the monoid of our automaton to a relation with (1,1). Clearly, there is a path from a starting state to a final state for these words. Therefore, they are recognized by the automaton and are part of the language. We formally define P in the next section.

Defining the set P

Recall the set of initial states I and the set of final states F of the automaton. We then define P as the set containing all relations with at least one pair of an initial and final state.

Definition 3.1.6. *We take the monoid* $(M, \cdot, 1)$ *of an automaton* $(S, \Sigma, I, F, \delta)$ *as defined in 3.1.2. We then define the subset* $P = \{(R|R \in M \wedge (r_1, r_2) \in R \wedge r_1 \in I \wedge r_2 \in F\}$.

Any relation that starts at an initial state, and ends at a final state implies that the automaton would have accepted the words that were mapped to this relation, and that these words are part of the language.

We can now prove that the monoid accepts the same language as the automaton. We prove this using two lemmas.

Lemma 3.1.7. *For the language L recognized by an NFA $(S, \Sigma, I, F, \delta)$, and the subset $P \subseteq M$ and homomorphism h of the monoid of the NFA as defined in 3.1.4 and 3.1.6, the following holds: $w \in L \rightarrow w \in h^{-1}(P)$.*

Proof. First we show that if $w \in L$ then $h(w) \in P$. Recall the definition of a language recognized by an automaton. This tells us that a word $w \in \Sigma^*$ is in L if $f \in \delta^*(w, i)$ for some $i \in I$ and $f \in F$.

If we look at the definition of h, we see that δ^* is used. It becomes immediately clear that if $w \in L$ then there will be a $(i, f) \in h(w)$ where i and f are initial and final states respectively. This obviously implies that $h(w) \in P$.

It becomes clear that $w \in h^{-1}(P)$ from the fact that $h(w) \in P$ and the definition of h^{-1} being the preimage. □

Lemma 3.1.8. *For the language L recognized by an NFA $(S, \Sigma, I, F, \delta)$, and the subset $P \subseteq M$ and homomorphism h of the monoid of the NFA as defined in 3.1.4 and 3.1.6, the following holds: $w \in h^{-1}(P) \rightarrow w \in L$.*

Proof. We know that h^{-1} is the preimage, and therefore that if $w \in h^{-1}(P)$ then it must be that $h(w) \in P$.

From the definition of h and P we know that $f \in \delta^*(w, i)$ for some $i \in I$ and $f \in F$. From the definition of the language recognized by an automaton, we see that $w \in L$. □

From these two lemmas we can conclude that we have $L = h^{-1}(P)$ for the language L recognized any finite monoid of an automaton and its subset P as defined in definition 3.1.6. Thus, if a language is regular, there is a finite monoid that recognizes the language.

3.2 From finite monoid to finite automaton

In the second part, we will provide an explanation for the proof of second implication of the theorem. We can describe this as follows.

Theorem 3.2.1. *Let L be a language over a finite alphabet Σ. If L is recognized by a finite monoid then it is regular.*

We prove the theorem by showing that we can create a deterministic finite automaton from this monoid that recognizes the same language. We first define such an automaton.

Definition 3.2.2 (automaton of a finite monoid)**.** Let us assume a monoid $(M, \cdot, 1)$, a finite set Σ, homomorphism $h : \Sigma^* \to M$, and subset $P \subseteq M$ such that the monoid recognizes a language $L = h^{-1}(P)$. We define the DFA $(S, \Sigma, I, F, \delta)$ of this monoid as follows:

- $S = M$
- $\Sigma = \Sigma$
- $I = 1_M$
- $F = P$
- $\delta(a, s) = s \cdot h(a)$

Additionally, we rewrite the definition of δ^* of a deterministic finite automaton for convenience, considering the previous definition.

Definition 3.2.3. For the automaton $(S, \Sigma, I, F, \delta)$ of a monoid as defined in 3.2.2, we may define the function δ^* as follows.
$\forall s \in S, \forall a \in \Sigma, \forall w \in \Sigma^*$

- $\delta^*(\varepsilon, s) = s$
- $\delta^*(a, s) = s \cdot h(a)$
- $\delta^*(aw, s) = \delta^*(w, s \cdot h(a))$

We show that for every word w there is a path from the initial state 1_M to the state $h(w)$. This would be the same as applying the second rule of δ^*, but with $s = 1_M$ and with a being a word instead of a letter.

Lemma 3.2.4. *We take a monoid $(M, \cdot, 1)$ and homomorphism h. Let the automaton of this monoid be $(S, \Sigma, I, F, \delta)$ as defined in definition 3.2.2. Then we have the following: $\forall w \in \Sigma^*, h(w) = \delta^*(w, 1)$*

Proof. We can prove this lemma by first proving another property. This property makes it possible to prove the lemma. The property is $\forall w, v \in \Sigma^*, \forall s \in S, \delta^*(w, s \cdot h(v)) = \delta^*(vw, s)$. We will prove this using induction on the length of v.

Base case:
We take $v = \varepsilon$.

$$\begin{aligned}\delta^*(w, s \cdot h(\varepsilon)) &= \delta^*(w, s \cdot 1) \\ &= \delta^*(w, s) \\ &= \delta^*(\varepsilon w, s)\end{aligned}$$

Induction step:
Let $w, v \in \Sigma^*$. Our induction hypothesis (IH) is the following:
$\delta^*(w, s \cdot h(v)) = \delta^*(vw, s)$.

We take some $a \in \Sigma$. We may write the following.

$$\begin{aligned}\delta^*(w, s \cdot h(va)) &= \delta^*(w, s \cdot h(v) \cdot h(a)) \\ &= \delta^*(aw, s \cdot h(v)) \\ &= \delta^*(vaw, s) \qquad \text{(IH)}\end{aligned}$$

This concludes the induction proof. Using this now proven property, which we call P, we prove the lemma. We have the following for all $w \in \Sigma^*$.

$$\begin{aligned}h(w) &= 1 \cdot h(w) \\ &= \delta^*(\varepsilon, 1 \cdot h(w)) \\ &= \delta^*(w\varepsilon, 1) \qquad \text{(P)} \\ &= \delta^*(w, 1)\end{aligned}$$

□

Using the property from the previous lemma 3.2.4, we now prove that the monoid and the automaton recognize the same language.

Lemma 3.2.5. *Take a finite monoid $(M, \cdot, 1)$, homomorphism h, subset $P \subseteq M$, and $L_M \subseteq \Sigma^*$. Let $L_M = h^{-1}(P)$ be the language recognized by M. Additionally we let L_A be the language recognized by the automaton $(S, \Sigma, I, F, \delta)$ of the monoid M defined as in definition 3.2.2. We then have $L_M = L_A$.*

Proof. Recall the definition of a language recognized by an automaton. For our language L_A this is the following.

$$L_A = \{w | f = \delta^*(w, i) \wedge f \in F \wedge i \in I\}$$

We may rewrite this as follows.

$$\begin{aligned} L_A &= \{w | f = \delta^*(w, i) \wedge f \in F \wedge i \in I\} \\ &= \{w | f = h(w) \wedge f \in F\} \\ &= \{w | h(w) \in P\} \end{aligned} \tag{3.2.4}$$

We now prove that if a word is in L_M then it is also in L_A, and that if a word is not in L_M then it is also not in L_A. This proves equivalence of the languages.

We take a word $w \in L_M$. As we have $L_M = h^{-1}(P)$, we have $h(w) \in P$. Considering the rewritten definition of L_A we have $w \in L_A$.

Now we take a word $w \notin L_M$. If for this word it holds that $h(w) \in P$, it would have been in L_M, which is contradictory. Therefore we have that $h(w) \notin P$ holds. Considering the rewritten definition of L_A we have $w \notin L_A$.

We can deduce that $L_M = L_A$, and thus that the languages recognized by the monoid and the automaton are the same. □

We can conclude that if a language is recognized by a finite monoid, there exists a finite automaton that recognizes the language, and thereby that the language is regular, proving theorem 3.2.1.

3.3 Conclusion of the proof

In part 1 we took an arbitrary non-deterministic finite automaton, and created a finite monoid based on it, as well as prove it is indeed a monoid in lemma 3.1.3. Additionally, we created a homomorphism from the automaton's alphabet to the monoid, and proved this is indeed a homomorphism in lemma 3.1.5. Finally, we define the subset P of the monoid, which we then use to prove the monoid and the automaton recognize the same language and thereby we also prove that if a language is regular, there is a monoid that recognizes the same language.

In part 2 we did the opposite, taking an arbitrary finite monoid, and creating a deterministic finite automaton based on it. We prove the useful property of lemma 3.2.4 which shows how we can easily rewrite usage of the homomorphism to the function δ^*. We used this property to prove that the monoid and the automaton recognize the same language, proving that if a language is recognized by a finite monoid, it is regular.

We can conclude that the proofs of part 1 and part 2 together prove theorem at the start of the chapter. To repeat this theorem, it is now proven that a language is regular if and only if it is recognized by a finite monoid.

Chapter 4

Groupoids and context-free languages

We have proven that finite monoids recognize regular languages by proving that finite automata and finite monoids recognize the same class of languages. We can also prove that finite groupoids, a structure similar to monoids, recognize context-free languages by proving that context-free grammars and finite groupoids recognize the same class of languages [1].

In the first section, we define what groupoids and context-free languages are and prove a useful property about groupoids. In the second and third sections, we use that property to prove the theorem about the recognition of context-free grammars by finite groupoids.

Additionally, in the fourth section we prove that if a language is recognized by a finite monoid, there exists a regular grammar that recognizes the same language. This proof is based on that in the second section but adapted to regular languages.

4.1 Groupoids

First, we define what groupoids and context-free languages are, alongside a few other necessary concepts. We start with defining groupoids.

Definition 4.1.1 (groupoid)**.** A groupoid is a tuple $(G, \cdot, 1)$ where G is a set, and $\cdot$ is a binary operator on G with 1 as the identity element. The following needs to hold $\forall q \in G$.

- $1 \cdot q = q \cdot 1 = q$

The structure we call a groupoid is often called an unital magma instead, and what is now commonly known as groupoid is something different. However, the source for the proof for finite groupoids recognizing context-free

languages used the name of groupoid, and we will use this name as well.

We now define a free monoid morphism, which maps a word $w \in \Sigma^*$ to a word of G^*, where Σ is some finite alphabet and G a groupoid.

Definition 4.1.2 (free monoid morphism)**.** We take a groupoid $(G, \cdot, 1)$ and some alphabet Σ. We may define the free monoid morphism as a monoid homomorphism $m : \Sigma^* \to G^*$ induced by $\hat{m} : \Sigma \to G$. This is to say, for any $w, v \in \Sigma^*$, we have $m(wv) = m(w)m(v)$ and for any $a \in \Sigma$ we have $m(a) = \hat{m}(a)$. Additionally, we define that $m(\varepsilon) = \varepsilon$.

Next, we recursively define binary trees.

Definition 4.1.3 (binary tree)**.** A binary tree T is a tuple (L, S, R). The elements L and R are either another binary tree or the empty set. The element S can be any element. With B_Σ we denote the set of all binary trees where for any tree $(L, S, R) \in B_\Sigma$, we have that $S \in \Sigma$.

We now define the groupoid word function.

Definition 4.1.4 (groupoid word function)**.** We take a groupoid $(G, \cdot, 1)$. We define the function $g : G^* \to \mathcal{P}(G)$ as the composition of two functions $g_1 : G^* \to \mathcal{P}(B_G)$ and $g_2 : \mathcal{P}(B_G) \to \mathcal{P}(G)$. We define these below.

First we define g_1 for all $a \in G$, $w \in G^*$ with $|w| \geq 2$.

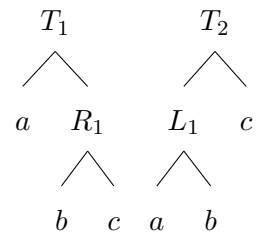
- $g_1(\varepsilon) = \{\emptyset, 1, \emptyset\}$
- $g_1(a) = \{(\emptyset, a, \emptyset)\}$
- $g_1(w) = \{(T_u, 1, T_v) | T_u \in g_1(u) \wedge T_v \in g_1(v) \wedge w = uv \wedge u \in G^* \backslash \{\varepsilon\} \wedge v \in G^* \backslash \{\varepsilon\}\}$

Finally we define g_2 for all Set in $\mathcal{P}(B_G)$. Additionally, we define a helper function $\hat{g}_2 : B_G \to G$ for any $L, R \in B_G$ and $S \in G$. Note that L and R cannot be the empty set per definition of B_G.

- $\hat{g}_2((\emptyset, S, \emptyset)) = S$
- $\hat{g}_2((L, S, R)) = \hat{g}_2(L) \cdot \hat{g}_2(R)$
- $\hat{g}_2((\emptyset, S, R)) = \hat{g}_2((L, S, \emptyset)) = S$
- $g_2(Set) = \{\hat{g}_2(T) | T \in Set\}$

We can see that the function yields the set of all groupoid elements that can be obtained by placing brackets in w, and evaluating the resulting expression using the binary operator of G. The function $\hat{g}_2$ has been defined for when for a tree (L, S, R) only L or R is the empty set. However, this is only for completeness and this case is never used.

Example 4.1.5. For example we may have a groupoid with $G = \{a, b, c, d, e, f, g\}$. We take a word $w = abc$ which is in G^*. First we use the function g_1 and create a set from abc with the two trees. We may visualize them as follows.



We can then evaluate T_1 by first applying $\hat{g}_2$ on R_1 which yields $b \cdot c$, and then applying $\hat{g}_2$ on T_1 which yields $a \cdot (b \cdot c)$. This can be similarly done for T_2, which yields $(a \cdot b) \cdot c$. We have $g(abc) = \{q_1, q_2\}$ with $q_1 = a \cdot (b \cdot c)$ and $q_2 = (a \cdot b) \cdot c$.

Using these morphisms we can have groupoids recognize languages. We define recognition of languages by groupoids as follows.

Definition 4.1.6 (language recognition by groupoids)**.** Take a groupoid G and alphabet Σ. We say G recognizes a language $L \subseteq \Sigma^*$ if there exists a morphism $m : \Sigma^* \to G^*$ and subset $P \subseteq G$ such that $L = \{w | g(m(w)) \cap P \neq \emptyset\}$.

We show an example to see how this recognition works and differs from recognition using monoids.

Example 4.1.7. We take a groupoid G, morphism $m : \Sigma^* \to G^*$, and subset $P \subseteq G$. For this groupoid we have abc is accepted if we have $m(a) \cdot (m(b) \cdot m(c)) \in P$ or we have $(m(a) \cdot m(b)) \cdot m(c) \in P$. Not both statements need to be valid, unlike recognition by monoids where these statements would be equivalent due to associativity.

We now define context-free languages.

Definition 4.1.8 (Context-free Grammar)**.** A context-free grammar is a tuple (N, Σ, R, S) where N is a finite set of non-terminal symbols, Σ is a finite alphabet with symbols, R is a finite set with production rules, and $S \in N$. The rules are of the form $A \to B$ where $A \in N$ and $B \in (N \cup \Sigma)^*$. We use a symbol $\overset{*}{\to}$, which is similar to $\to$ but it shows the step consists out of one or more steps. For these rules we have the following for $A \in N$ and $B, C, D \in (N \cup \Sigma)^*$.

- $A \to B$ implies $CAD \overset{*}{\to} CBD$
- $A \overset{*}{\to} B$, $B \overset{*}{\to} C$ implies $A \overset{*}{\to} C$
- $B \overset{*}{\to} B$

We then define the language recognized or generated by this grammar as $L = \{w \in \Sigma^* | S \overset{*}{\to} w\}$.

Groupoid Property

We now prove a few interesting properties which hold for any groupoid and a morphism $m : \Sigma^* \to G^*$. We will use these properties in the other sections.

Lemma 4.1.9 (groupoid-property-left)**.** *For any groupoid G, alphabet Σ and a morphism $m : \Sigma^* \to G^*$, we have for all $p, q \in G, u, v \in \Sigma^* \backslash \{\varepsilon\}$ such that, $p \in g(m(u))$ and $q \in g(m(v))$ the following: $(p \cdot q) \in g(m(uv))$.*

Proof. Let $x, y \in G^*$ such that $m(u) = x, m(v) = y$. Since m is a homomorphism, we have that $m(uv) = xy$.

From the definition of g_1 we know that $g_1(xy) = \{(T_a, 1, T_b) | T_a \in g_1(a) \wedge T_b \in g_1(b) \wedge xy = ab \wedge a \in G^* \backslash \{\varepsilon\} \wedge b \in G^* \backslash \{\varepsilon\}\}$. We let T_{xy} be in the subset of $g_1(xy)$ where $a = x$ and $b = y$.

We know that $p \in g(m(u))$ and therefore that there exists a tree $T_x \in g_1(m(u))$ such that $\hat{g}_2(T_x) = p$. Similarly, we have a tree for q. This leads to there existing two trees $T_x \in g_1(m(u)), T_y \in g_1(m(v))$ such that $\hat{g}_2(T_x) = p$ and $\hat{g}_2(T_y) = q$.

From the definition of g_2 we know that $g_2(\{T_{xy}\}) = \{\hat{g}_2(T_{xy})\} = \{\hat{g}_2(T_x) \cdot \hat{g}_2(T_y)\} = \{p \cdot q\}$. We know that $xy = m(uv)$, and we can thus conclude that $(p \cdot q) \in g(m(uv))$.

□

Lemma 4.1.10 (groupoid-property-right)**.** *For any groupoid G, alphabet Σ and a morphism $m : \Sigma^* \to G^*$, we have for all $r \in G, w \in \Sigma^*$ such that $r \in g(m(w))$ and $|w| \geq 2$ the following. There exist elements $p, q \in G, u, v \in \Sigma^* \backslash \{\varepsilon\}$ such that $p \cdot q = r$ and $w = uv$, and that $p \in g(m(u))$ and $q \in g(m(v))$.*

Proof. There has to be a tree T_r such that $\hat{g}_2(T_r) = r$. Additionally, we need to have $T_r \in g_1(m(w))$. This is because $r \in g(m(w))$ would be false otherwise.

As $|w| \geq 2$ we use the third case of g_1 and have $g_1(m(w)) = \{(T_u, 1, T_v) | T_u \in g_1(m(u)) \wedge T_v \in g_1(m(v)) \wedge m(w) = m(uv) \wedge m(u) \in G^* \backslash \{\varepsilon\} \wedge m(v) \in G^* \backslash \{\varepsilon\}\}$. We have that T_r is one of these trees in $g_1(m(w))$ for some $u, v \in \Sigma^* \backslash \{\varepsilon\}$ such that $w = uv$.

We then have $\hat{g}_2(T_r) = \hat{g}_2(T_u) \cdot \hat{g}_2(T_v)$ for some $T_u \in g_1(m(u))$ and $T_v \in g_1(m(v))$. If we call the results of $\hat{g}_2(T_u)$ and $\hat{g}_2(T_v)$ elements p and q respectively, we get $r = p \cdot q$. We also have for p that $p \in g(m(u))$ since $p \in g_2(g_1(m(u)))$, and similarly for q, which finishes the proof. □

The theorem

In the next sections we will prove the equivalence of context-free grammars and finite groupoids. For the proofs we follow the proof outline as given in [1, Lemma 3.1]. Formally we can state the theorem as follows.

Theorem 4.1.11. *A language L over a finite alphabet Σ is context-free if and only if L is recognized by a finite groupoid G as defined in 4.1.6.*

In the first section this involves showing we can create a context-free grammar for any finite groupoid, and that this grammar generates the same language that the groupoid recognizes. In the second section we do the opposite, creating a finite groupoid for any context-free grammar.

4.2 From finite groupoid to context-free grammar

In this section, we let L be a language over a finite alphabet Σ. We will then provide a proof for the following statement.

Theorem 4.2.1. *Let L be a language over the finite alphabet Σ. We have that if the language L is recognized by a finite groupoid then L is context-free.*

We accomplish this by creating a context-free grammar and showing that this grammar generates L.

Defining a grammar from a finite groupoid

We first define a context-free grammar which we can use later.

Definition 4.2.2. Let $(G, \cdot, 1)$ be a finite groupoid and Σ a finite alphabet. Let $P \subseteq G$ and morphism $m : \Sigma^* \to G^*$ be such that $L = \{w \in \Sigma^* | g(m(w)) \cap P \neq \emptyset\}$. We construct a context-free grammar (N, Σ, R, S) as follows.

$$
\begin{aligned}
N &= \{S\} \cup G \\
R &= \{m(a) \to a | a \in \Sigma\} && (1) \\
&\quad \cup \{S \to q | q \in P\} && (2) \\
&\quad \cup \{1 \to \varepsilon\} && (3) \\
&\quad \cup \{q \to ps | p, s \in G \wedge p \cdot s = q\} && (4)
\end{aligned}
$$

In rule (1) we may assume $m(a) \in G$ since $a \in \Sigma$ and by definition of m it will therefore map to only one element in G. Please note that we assume the sets do not overlap and in other words assume $\{S\} \cap G = \Sigma \cap G = \{S\} \cap \Sigma = \emptyset$. We now prove this grammar is indeed context-free.

Lemma 4.2.3. *The grammar of a finite groupoid $(G, \cdot, 1)$ as defined in 4.2.2, is indeed a context-free grammar.*

Proof. We see that N is finite since G is finite. The same holds for Σ. The set R is also finite, since Σ and P are finite so there are a finite amount of rules of type (1) and (2). There is only one rule of type (3). Additionally $G \times G$ is finite, so there are finite amount of rules of type (4). □

Generating words

We now prove an interesting property of any non-terminal when we know it generates a certain word.

Lemma 4.2.4. *We take a groupoid $(G, \cdot, 1)$, morphism m, subset $P \subseteq G$, and its grammar (N, Σ, R, S) as defined in definition 4.2.2. We have for all $w \in \Sigma^*$, if $q_w \in g(m(w))$ then $q_w \xrightarrow{*} w$.*

Proof. We prove this using induction on the length of w. We use a property that requires words longer than one letter, so we prove two base cases. We have one case for the empty word and one for words with one letter.

Base cases:
We have $w = \varepsilon$. We know that $g(m(\varepsilon)) = \{1\}$. We simply apply the defined rule.

$$1 \to \varepsilon$$

For our second base case we have $w = a, a \in \Sigma$. We know that $m(a)$ yields one element in G, since a is a letter. We simply apply the defined rule.

$$m(a) \to a$$

Induction step:
We have $q_w \in g(m(av))$ where $a \in \Sigma$, $v \in \Sigma^*$ and $|v| \geq 2$. Our induction hypothesis is the following: for any word $u \in \Sigma^*$ where $|u| < |w|$ we have for any $q \in g(m(u))$ that $q \xrightarrow{*} u$.

According to lemma 4.1.10 we may assume some q_x and q_y exist such that $q_w = q_x \cdot q_y$. Therefore, we can apply the defined grammar rule.

$$q_w \to q_x q_y$$

Additionally, we also know from 4.1.10 that $q_x \in g(m(x)), q_y \in g(m(y))$ and $w = xy$. Since we know neither x nor y are the empty word, x and y are words shorter than w. therefore we may apply the induction hypothesis twice.

$$q_w \to q_x q_y \xrightarrow{*} x q_y \xrightarrow{*} xy$$

□

Lemma 4.2.5. *For a groupoid $(G, \cdot, 1)$, morphism m, subset $P \subseteq G$, and its grammar (N, Σ, R, S) as defined in definition 4.2.2, we have for all $q_w \in N \backslash \{S\}, w \in \Sigma^*$ such that $q_w \xrightarrow{*} w$ the following: $q_w \in g(m(w))$.*

We do not prove this lemma here as we consider it out of scope for this thesis. We may combine the two lemmas 4.2.4 and 4.2.5 into one corollary.

Corollary 4.2.6. *We take a groupoid $(G, \cdot, 1)$, morphism m, subset $P \subseteq G$, and its grammar (N, Σ, R, S) as defined in definition 4.2.2. We have for all $q_w \in N \backslash \{S\}, w \in \Sigma^*$ that $q_w \xrightarrow{*} w$ if and only if $q_w \in g(m(w))$.*

Finishing the proof

We can now prove that the finite groupoid and our defined grammar recognize the same language.

Lemma 4.2.7. *We take a groupoid $(G, \cdot, 1)$, morphism m, subset $P \subseteq G$ and the language $L \subseteq \Sigma^*$ recognized by it. We also define its grammar (N, Σ, R, S) as in 4.2.2. We have a word $w \in L$ if and only if $S \xrightarrow{*} w$.*

Proof. We take a word $w \in L$. We know that $g(m(w)) \cap P \neq \emptyset$. Therefore we have an element $q_w \in P$, $q_w \in g(m(w))$ and thereby $S \to q_w$. Together with corollary 4.2.6 we have the following.

$$S \to q_w \xrightarrow{*} w$$

Now we take a word such that $S \xrightarrow{*} w$. We apply the only type of rule possible for S and get $S \to q_w \xrightarrow{*} w$ for some $q_w \in N$. From corollary 4.2.6 we know that $q_w \in g(m(w))$. From the definition of the grammar we also know $q_w \in P$. We can see that we have $w \in L$. Thus the groupoid recognizes the same language that the grammar generates. □

We have now proven that if a language is recognized by a finite groupoid then this language is context-free.

4.3 From context-free grammar to finite groupoid

In this section, we let L be a language over a finite alphabet Σ. We will then provide a proof for the following statement.

Theorem 4.3.1. *Let L be a language over a finite alphabet Σ. We have that if the language L is context-free then L is recognized by a finite groupoid.*

We accomplish this by creating a finite groupoid and morphism, and showing that this recognizes the same language as the grammar for L generates. We also use the following theorem.

Theorem 4.3.2. *Any context-free grammar $D = (N, \Sigma, R, S)$ can be changed such that R contains only rules that are of one of the following forms, for $A, B, C \in N$, and $a \in \Sigma$.*

- $A \to a$
- $A \to BC$
- $S \to \varepsilon$
- $S \to A$

There may be no rules where the right side is the same.

This has already been proven in [5]. We choose not to prove this theorem here, as we consider this out of scope for this thesis,

Defining groupoid from a context-free grammar

We define a groupoid G using a context-free grammar as follows.

Definition 4.3.3. Assume a context-free grammar (N, Σ, R, S) which is in the form as described in 4.3.2. We then define the groupoid of this grammar as $G = (N \backslash \{S\}) \cup \{1, \$\}$. We define the binary operator as follows for any $p, q, r \in G$.

- $1 \cdot p = p \cdot 1 = p$
- $\$ \cdot p = p \cdot \$ = \$$
- $p \cdot q = r$ if and only if $(r \to pq) \in R$.
- $p \cdot q = \$$ if the other rules have not defined it.

We show this is indeed a finite groupoid.

Lemma 4.3.4. *We take a context-free grammar (N, Σ, R, S) in the form as described in 4.3.2. Then the groupoid of this grammar as defined in 4.3.3 is indeed a finite groupoid.*

Proof. We know that N is finite, and therefore G is finite too. Additionally, 1 is clearly the identity element by definition.

We still need to make sure the binary operator is not defined twice for the same pair of elements. We know that the grammar contains no rules where the right side is the same. Therefore for some $p, q \in G$ there will be at most one rule $(r \to pq) \in R$ and thus also the binary operator for $p \cdot q$ will never be defined twice.

Finally, we see that the binary operator is always defined since any pair $p, q \in G$ for which it is not defined by the grammar or the identity element is defined by $p \cdot q = \$$. □

We now define a set $P \subseteq G$.

Definition 4.3.5. We take a context-free grammar (N, Σ, R, S) and its groupoid $(G, \cdot, 1)$. We then define the set $P = \{q | (S \to q) \in R\}$ if $\varepsilon \notin L$ and otherwise $P = \{q | (S \to q) \in R\} \cup \{1\}$.

This set P will be used with a morphism to make the groupoid recognize the language. We define the morphism.

Definition 4.3.6. We take a context-free grammar (N, Σ, R, S) and its groupoid G. We define the morphism of G as $m : \Sigma^* \to G^*$ as induced by $m(a) = q$ if and only if $(q \to a) \in R$ for any $a \in \Sigma$. Additionally, $m(a) = \$$ if and only if $(q \to a) \notin R$.

Combining elements from the groupoid

We have now defined the groupoid of a context-free grammar. We now prove a few essential properties.

Lemma 4.3.7. *For any grammar (N, Σ, R, S), its groupoid $(G, \cdot, 1)$ and morphism m as defined in definition 4.3.3 and 4.3.6, we have for any $q_w \in N \backslash \{S\}, w \in \Sigma^*$ where $w \neq \varepsilon$ that if $q_w \in g(m(w))$ then $q_w \xrightarrow{*} w$ holds.*

Proof. We prove this using induction on the length of w.

Base case:
We have $q \in g(m(a))$ where $a \in \Sigma$. From the definition of our morphism we know that $(q \to a) \in R$.

Induction step:
We have $q_w \in g(m(w))$ where $w \in \Sigma^*$. Our induction hypothesis is the following: for any $q_v \in N \backslash \{S\}$ and $v \in \Sigma^*$ where $|v| < |w|$ we have $q_v \xrightarrow{*} v$ if $q_v \in g(m(v))$.

From lemma 4.1.10 we know that there exist $q_x, q_y \in G$ such that $q_w = q_x \cdot q_y$. From the definition of our groupoid this may only be the case if $(q_w \to q_x q_y) \in R$.

Additionally, we know from lemma 4.1.10 that $q_x \in g(m(x)), q_y \in g(m(y))$ where $w = xy$ and neither x nor y are the empty word. Since x and y are words shorter than w we may apply the induction hypothesis twice.

$$q_w \to q_x q_y \xrightarrow{*} x q_y \xrightarrow{*} xy$$

Since $w = xy$, this proves the lemma.

□

Lemma 4.3.8. *For any grammar (N, Σ, R, S), its groupoid $(G, \cdot, 1)$ and morphism m as defined in definition 4.3.3 and 4.3.6 we have for any $q_w \in N \backslash \{S\}, w \in \Sigma^*$ where $w \neq \varepsilon$ that if $q_w \xrightarrow{*} w$ then $q_w \in g(m(w))$ holds.*

We do not prove this lemma here as we consider it out of scope for this thesis. We can combine the previous two lemmas into one corollary.

Corollary 4.3.9. *For any grammar (N, Σ, R, S), its groupoid $(G, \cdot, 1)$ and morphism m as defined in definition 4.3.3 and 4.3.6 we have for every $w \in \Sigma^*$ that is not ε, and $q \in G$, the following: $q_w \xrightarrow{*} w$ if and only if $q_w \in g(m(w))$.*

Finishing the proof

We may now prove that the context-free grammar and our defined finite groupoid recognize the same language.

Lemma 4.3.10. *For any grammar (N, Σ, R, S), its groupoid $(G, \cdot, 1)$ and morphism m as defined in definition 4.3.3 and 4.3.6, we have $S \xrightarrow{*} w$ if and only if $g(m(w)) \cap P \neq \emptyset$.*

Proof. First, we look at the special case $w = \varepsilon$. We know there is only one possible rule that leads to ε. It is defined that $S \to \varepsilon$ if and only if $1 \in P$ in definition 4.3.5. Thus the lemma holds for $w = \varepsilon$

We now assume $S \xrightarrow{*} w$ for any $w \in \Sigma^*$ where $|w| \geq 1$. Applying the only allowed rule for S that does not lead to the empty word we have $S \to q \xrightarrow{*} w$ for some $q \in N$. We may apply the corollary 4.3.9, and get $q \in g(m(w))$.

Additionally, from the definition of the groupoid in definition 4.3.3 we have $q \in P$.

Finally, we may assume $q \in g(m(w))$ and $q \in P$. This may only be the case if $(S \to q) \in R$. Additionally, from the corollary 4.3.9 we know that $q \xrightarrow{*} w$. We combine these two rules and get $S \xrightarrow{*} w$. □

We have now proven that every context-free language is recognized by some finite groupoid.

The theorem proved in the previous section and the theorem proved in this section together prove theorem 4.1.11. This means that finite groupoids recognize exactly context-free languages.

4.4 Regular grammars and finite monoids

In the previous sections we proved the equivalence of context-free grammars and finite groupoids. We could also prove that finite monoids recognize regular languages by using regular grammars. This differs from the proof in chapter 3 where we used finite automata instead of grammars. We first define regular grammars.

Definition 4.4.1 (Regular Grammar)**.** A grammar (N, Σ, R, S) is regular if the rules are in one of the following forms with $q, p \in N$, $a \in \Sigma$.

- $q \to ap$
- $q \to a$
- $q \to \varepsilon$

We do not fully prove that finite monoids recognize regular languages using regular grammars, even though it should be possible since regular grammars generate exactly the regular languages, since we consider this out of scope for this thesis. We only prove the direction of going from a finite monoid to a regular grammar, which we will show in the next few sections.

From finite monoid to regular grammar

In this section we let L be a language over a finite alphabet Σ. We will then provide a proof for the following statement.

Theorem 4.4.2. *Let L be a language over a finite set Σ. Then we have that if the language L is recognized by a finite monoid, then L can be produced by a regular grammar.*

We will prove this by defining a regular grammar for any finite monoid. We later show this regular grammar generates the same language as the finite monoid recognizes.

Defining the grammar

We first define a regular grammar for any finite monoid that recognizes some language.

Definition 4.4.3. Let $(M, \cdot, 1)$ be a finite monoid. Let $P \subseteq M$ and homomorphism m be such that $L = m^{-1}(P)$. We construct its regular grammar (N, Σ, R, S).

$$
\begin{aligned}
N &= S \cup M \\
\Sigma &= \Sigma \\
R &= \{S \to q | q \in P\} && (1) \\
&\quad \cup \{1 \to \varepsilon\} && (2) \\
&\quad \cup \{q \to ap | p \in M \wedge m(a) \cdot p = q\} && (3)
\end{aligned}
$$

We note that sets $\{S\}, M$, and Σ are disjoint. In other words, we have that $\{S\} \cap M = \{S\} \cap \Sigma = \Sigma \cap M = \emptyset$.

Generating words

We have defined the grammar for a finite monoid that recognizes a language. We now prove a few essential properties.

Lemma 4.4.4. *We take a monoid $(M, \cdot, 1)$, homomorphism m and its grammar (N, Σ, R, S) as defined in 4.4.2. For any word $w \in \Sigma^*$ and $m(w) \in N \backslash \{S\}$, we have $m(w) \xrightarrow{*} w$.*

Proof. We prove using induction on the length of w.

Base case:
We have $w = \varepsilon$. We know by definition of a homomorphism that $m(\varepsilon) = 1$ and from rule (2) that $1 \to \varepsilon$, which fulfills the base case.

Induction Step:
We have $w = av$ with $a \in \Sigma$. Our induction hypothesis (IH) is the following: for any $v \in \Sigma^*$ and $m(v)$ where $|v| < |w|$, we have $m(v) \xrightarrow{*} v$.

We may use rule (3) and get $m(av) \to am(v)$. We may then apply the induction hypothesis.

$$m(av) \to am(v) \xrightarrow{*} av$$

□

Lemma 4.4.5. *We take a monoid $(M, \cdot, 1)$, homomorphism m and its grammar (N, Σ, R, S) as defined in 4.4.2. Let $q, r \in N \backslash \{S\}$. If $q \xrightarrow{*} r$ then $q = r$.*

We do not prove this lemma here as we consider it out of scope for this thesis. We use it to support the next lemma.

Lemma 4.4.6. *We take a monoid $(M, \cdot, 1)$, homomorphism m and its grammar (N, Σ, R, S) as defined in 4.4.2. Taking $w \in \Sigma^*$ and $q_w \in N \backslash \{S\}$ such that $q_w \xrightarrow{*} w$, we have that $q_w = m(w)$.*

Proof. We prove using induction on the length of w.

Base case:
We have $w = \varepsilon$. From rule (2) we get $q_w \xrightarrow{*} 1 \to \varepsilon$. Rule (1) cannot be applied since we assume $q_w \neq S$ nor can rule (3) since this generates a letter and another non-terminal. By lemma 4.4.5 we have $q_w = 1$. Finally, we know by definition of a homomorphism that $m(\varepsilon) = 1 = q_w$, which fulfills the base case.

Induction Step:
We take $w = av$ and have $q_w \xrightarrow{*} av$ with $a \in \Sigma$. Our induction hypothesis (IH) is the following: for all $p \in N$ and $v \in \Sigma^*$ where $p \xrightarrow{*} v$ and $|v| < |w|$, we have $p = m(v)$.

We cannot apply rule (1) since we know $q_w \neq S$ nor rule (2) since this leads to the empty word, but w contains at least one letter. Therefore, we have to apply rule (3). For some $b \in \Sigma$ and $p \in M$ where $m(b) \cdot p = q_w$ we have the following.

$$q_w \to bp \xrightarrow{*} av$$

Since the first letter of w is a, we need to have that $b = a$. Therefore we have $q_w \to ap \xrightarrow{*} av$ and with this also that $p \xrightarrow{*} v$. We may apply the induction hypothesis and have $p = m(v)$. From this and the condition of the rule applied earlier we have $m(a) \cdot p = m(a) \cdot m(v) = q_w$. We know m is a homomorphism, thus we may conclude $m(a) \cdot m(v) = m(av) = q_w$. □

We may combine the lemmas 4.4.4 and 4.4.6 into one corollary.

Corollary 4.4.7. *We take a monoid $(M, \cdot, 1)$, homomorphism m and its grammar (N, Σ, R, S) as defined in 4.4.2. Taking some word $w \in \Sigma^*$ and $q \in N \backslash \{S\}$, we have that $q \xrightarrow{*} w$ if and only if $q = m(w)$.*

Finishing the proof

We can now prove that the finite monoid and our defined regular grammar recognize the same language.

Lemma 4.4.8. *We take a monoid $(M, \cdot, 1)$, homomorphism m, subset P, the language L recognized by it, and the monoid's grammar (N, Σ, R, S) as defined in 4.4.2. We have that $S \xrightarrow{*} w$ if and only if $w \in L$.*

Proof. We take a word $w \in L$. We know that $m(w) \in P$. By rule (1) we have $S \to m(w)$. Together with corollary 4.4.7 we have the following.

$$S \to m(w) \xrightarrow{*} w$$

Now we take $S \xrightarrow{*} w$ for some $w \in \Sigma^*$. We take the only possible rule from S and we get $S \to q \xrightarrow{*} w$ where $q \in N$ and $q \in P$. From corollary 4.4.7 we get $q = m(w)$. We have that $m(w) \in P$ and thus $w \in L$. □

We have now proven that for any monoid that recognizes a language, we can create a regular grammar that generates the same language.

This proof partially resembles the proof of section 4.2, but in this section we considered finite monoids and regular grammars instead.

Chapter 5

Related Work

The original source of the proof for the theorem of finite monoids recognizing regular languages [12] is very old. They prove very compactly and also prove other theorems in the paper. This may be considered difficult to understand.

The theorem of finite monoid recognizing regular languages can be very useful. Algebraic representations for regular languages such as monoids can be used for efficient algorithms such as can be seen by several algorithms on strings and trees implemented using monoids [2] and an algorithm for the equational theory of binary relations [11].

Several textbooks also mention the theorem and related concepts [6, 4]. In [4, Proposition 10.1] they very briefly prove the equivalence of recognition of a set using finite monoids and two other statements, namely the set being recognizable and the syntactic monoid of this set being finite. In [6], they mention recognition of a set using finite monoids, and also that this is equivalent to using finite automata, but do not show any proof.

The original source of the proof for the theorem of finite groupoids recognizing context-free languages [1] describes the proof nicely, but is very brief, leaving out some steps. This theorem was already partially implicit in Valiant's work [14].

In the same paper as the proof of groupoids recognizing context-free languages, they consider two extensions to Barrington's M-program model. This is a model of computation that has been studied a lot in relation to the structure of a certain complexity class NC^1. An M-program loosly consists of sequences of simple instructions, one for each input length, which translates an input string into a word over a monoid, and then evaluates it in the monoid which determines acceptance of the input. The extensions consider using groupoids instead of monoids [1].

A theorem about aperiodic monoids and star-free languages which is similar to the previous mentioned theorems has been proven in [12]. However, similar to the other theorems it is not very legible. This theorem has already been explained more thoroughly by Pippenger [10] who followed the original proof.

Chapter 6

Conclusions

In the first part of this thesis, we worked out and looked at an elaborate proof for the theorem of finite monoids recognizing regular languages. The proof consists of two directions. First, we worked out the proof where we create a finite monoid assuming a finite automaton and showed that this monoid recognizes the same language. Then, we worked out the proof where we assume a finite monoid and create a finite automaton and similarly showed this automaton recognizes the same language.

In the second part, we worked out and looked at an elaborate proof for the theorem of finite groupoids recognizing context-free languages. The proof consists of two directions. To start, we worked out the proof where we create a context-free grammar assuming a finite groupoid and showed that the groupoid recognizes the same language. Then, we worked out the proof where we assume a finite groupoid and create a context-free grammar and similarly showed this grammar generates the same language.

We can see some similarities between the proofs of the two theorems. The algebraic structures are very similar, and the proofs both show they recognize certain classes of languages, namely the regular languages and context-free languages respectively. The proofs differ in the sense that the structures we assume to recognize a class of languages is different. In the first proof, automata are used but in the second proof grammars are used.

In the third and final part, we created an alternative proof for part of the theorem of finite monoids recognizing regular languages. We did this by using the fact regular grammars recognize regular languages instead of the fact finite automata recognize them. To be precise, we proved the direction of creating a regular grammar from a finite monoid. The new proof was inspired by the proof in the second part, but instead concerns regular grammars and regular languages. While the theorem had already been proven,

this alternate proof is interesting as it does not appear to be very different in difficulty than the other proof and it confirms the theorem is provable in multiple ways.

In this thesis we did not discuss a proof for the direction of creating a finite monoid assuming a regular grammar. Since regular grammars describe exactly the regular languages, this direction should be possible to prove. The proof for this part of the theorem may be considered in future work. Additionally, we left out the proofs for a few lemmas in chapter 4 which are needed to fully finish the proof. These may also be considered in future work.

Bibliography

[1] François Bédard, François Lemieux, and Pierre McKenzie. Extensions to barrington's m-program model. *Theoretical Computer Science*, 107(1):31–61, 1993.

[2] Mikołaj Bojańczyk. Algorithms for regular languages that use algebra. *SIGMOD Rec.*, 41(2):5–14, August 2012.

[3] Janusz A Brzozowski and Michael Yoeli. Digital networks. *(No Title)*, 1976.

[4] Samuel Eilenberg. *Automata, Languages, and Machines, part 1.* Academic Press, 1974.

[5] Michael A Harrison. *Introduction to formal language theory.* Addison-Wesley Longman Publishing Co., Inc., 1978.

[6] Reuben Thomas Jacques Sakorovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[7] S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956.

[8] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[9] M. Lothaire. *Combinatorics on words.* Cambridge University Press, 1997.

[10] Nicholas Pippenger. *Theories of computability.* Cambridge University Press, 1997.

[11] Damien Pous and Jana Wagemaker. Completeness theorems for kleene algebra with tests and top, 2024.

[12] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965.

[13] Imre Simon. Piecewise testable events,(h. brakhage ed.), automata theory and formal languages, lncs. 33, 1975.

[14] Leslie Valiant. General context-free recognition in less than cubic time. 1974.