Bachelor Thesis
Computing Science

Radboud University

Closure Properties of Nominal Languages under Substitutions

Author:
Menno Bartels
S1007797

First assessor:
dr. J.C. Rot
jrot@cs.ru.nl

Second assessor:
prof. dr. J.H. Geuvers
h.geuvers@cs.ru.nl

March 26, 2021

Abstract

Nominal sets are useful for computation over infinite input alphabets. We will focus on nominal automata theory, which uses nominal sets. We investigate the closure properties of nominal set under so-called substitutions. To do this we define theory for nominal sets, following Klin et al. (2014), and give proofs for a closure result and a Myhill-Nerode style theorem. We also give a counterexample which shows that being recognized by a nominal DFA, is not closed under nominal substitutions.

Contents

Chapter 1

Introduction

In theoretical computing science, automata theory studies abstract computational machine [7, 8]. A computer program can be abstracted into a state diagram, which represents the behaviour of the program. A state diagram often consists of circles, depicting the states in which a program can be, and arrows, which show on what input the program moves between states. Such a state diagram is often called an automaton. In automata theory, we define state diagrams in a formal way, so that we are able to formally reason about them. It is also possible to investigate the behaviour of a computer program. From the results of such investigations, it is (in some cases) possible to construct an automaton that exactly describes the behaviour of the program. This is known as automata learning, which can be seen as a research field on its own [1, 6].

For computer programs, the state diagrams we want to make are not always finite. Because of this we want to introduce structure, so that we can still reason about infinite sets. This is where nominal sets come into play [2]. With the properties of nominal sets, we are able to group elements in such a way that each group can be distinguished in a similar way. If there is a finite amount of these groups, we can then create state diagrams, using these group. By making these groups, which will later be called orbit, we have found a way to depict an infinite state diagram, in a finite way. We will also see, that in some cases it is not possible to make a finite amount of groups.

One of the big goals of this thesis is to understand nominal languages, and investigate their behaviour when applying substitutions. To be able to achieve this goal, we have to introduce theory and prove theorems. This thesis combines pure mathematics with fundamental theoretical computing science, which makes it quite theoretical. With this in mind, the theory is presented in such a way that co-students, should be able to understand.

In this thesis we work out a proof of a Myhill-Nerode style theorem, for regular and for nominal languages, which we found in the work of Klin et al. [2]. Loc cit. we take the theorem that is given, and reformulate it into the framework of this thesis. We also prove that nominal languages are closed under substitution, and give a counterexample which shows that being recognized by a nominal deterministic automaton is not equivalent to having orbit finite derivatives.

In Chapter 2 we will start by giving basic definitions and theorems about regular languages and deterministic automata. Then, in Chapter 3 we will introduce some group theory and we will define how substitutions on languages behave. Also we will define nominal sets, and show some properties that these sets have. In Chapter 4 we will combine our knowledge of automata, group theory and nominal sets to introduce nominal automata. Lastly, in Chapter 5 we will investigate how nominal automata behave when we apply the earlier defined substitutions, and prove that nominal sets are closed under substitutions.

Chapter 2

Automata theory

In this chapter we will give some definitions and basic theorems related to automata theory. These include elementary definition, which we include, to prevent ambiguity about notation. Similar definitions can be found in textbooks such as [7] and [8].

2.1 Deterministic finite automata

Definition 2.1.1. A *deterministic finite automaton* (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite nonempty *set of states*, Σ is the finite nonempty *input alphabet*, $\delta : Q \times \Sigma \to Q$ is *the transition function*, $q_0 \in Q$ is *the intitial state*, and $F \subseteq Q$ is the set of *final states*. The set of final states is also sometimes called the set of *accepting states*.

Definition 2.1.2. A *language* is a subset $L \subseteq \Sigma^*$, where Σ is a finite nonempty alphabet.

An automaton can recognize a language. This means that given word w and a language L, the automaton can give the answer to the question "is w a word that is in the language L?". By applying the transition function on a word we can check if the resulting state is a final state. If this is the case, that means that $w \in L$. The transition function applied on a word is of the form $\delta^* : Q \times \Sigma^* \to Q$. By iterating δ on a word we get the function δ^*. Suppose $xw \in \Sigma^*$ is a word, where we explicitly write the x in front of w, then δ^* is defined as:

$$\delta^*(q_i, xw) = \delta^*(\delta(q_i, x), w)$$

and

$$\delta^*(q_i, \lambda) = q_i$$

where $x \in \Sigma$ and $w \in \Sigma^*$.

The class of languages that are recognized by DFA's are called the *regular languages*. A language that can not be recognized by a DFA is called a *non-regular language*.

Definition 2.1.3. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton. For $q \in Q$, we define *the language of state* q as

$$\mathcal{L}(q) = \{w \mid \delta^*(q, w) \in F\}$$

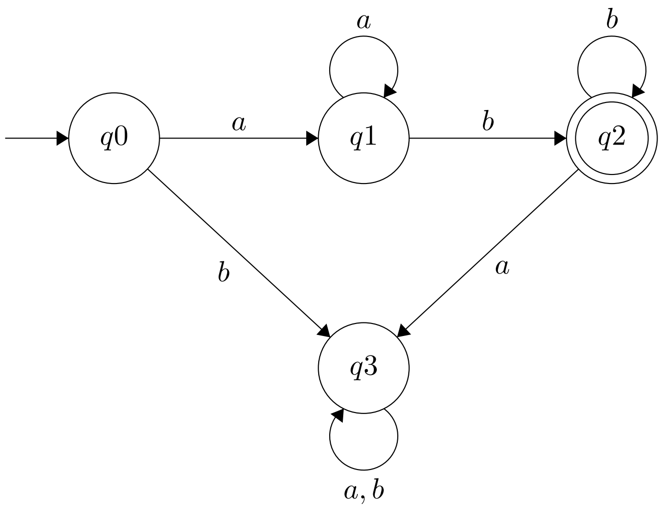
where the the language of the machine M is equal to $\mathcal{L}(q_0)$.

To give a better feel for what regular languages are, we will look at some examples of regular languages, and also give the automaton that recognized the language. We will also look at an example of a non-regular language.

Example 2.1.4. The empty language $L = \emptyset$ is a regular language. This language is recognized by the automaton with only one state. This state is the initial state.

A DFA that recognizes a regular language can be drawn as a state diagram. This is done as follows: depict the states as circles, with state names in them, denote the initial state with an in-going arrow, depict the final state with a double circle, and display the behaviour of δ with arrows. We will show an example of a state diagram in the next example.

Example 2.1.5. The language $L = \mathcal{L}(a^+b^+)$ are all the words of the form uv, where u contains at least one a and no b's, and v contains at least one b and no a's. The language L is regular because we can construct a DFA M such that M recognizes L. A state diagram of M could look like this:



We can now investigate the structure of $M = (Q, \Sigma, \delta, q_0, F)$. The state space is $Q = \{q_0, q_1.q_2, q_3\}$. We see that the alphabet is $\Sigma = \{a, b\}$ since this is also the alphabet of L. Then, δ is defined by the following table:

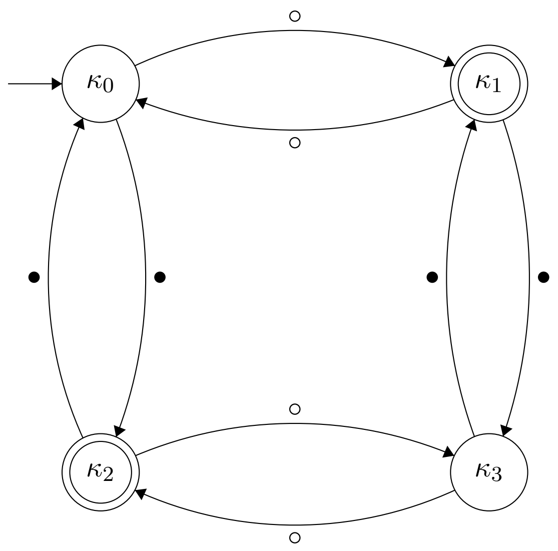
$\delta(q_i, x)$	q_0	q_1	q_2	q_3
a	q_1	q_1	q_3	q_3
b	q_3	q_2	q_2	q_3

The initial state q_0 is conveniently named q_0, and the set of final states $F = \{q_2\}$. An example of a word that is in L is the word *aab*. This word is in L since applying the transition function δ^* gives:

$$\begin{aligned}
\delta^*(q_0, aab) &= \delta^*(\delta(q_0, a), ab) \\
&= \delta^*(q_1, ab) \\
&= \delta^*(\delta(q_1, a), b) \\
&= \delta^*(q_1, b) \\
&= \delta^*(\delta(q_1, b), \lambda) \\
&= \delta^*(q_2, \lambda) \\
&= q_2
\end{aligned}$$

Since q_2 is a final state, we may conclude that the word *aab* is in the language L. An example of a word that is not in L is the word *abba*, since $\delta^*(q_0, abba) = q_3$, which is not a final state

Example 2.1.6. Let $\Sigma = \{\circ, \bullet\}$. We now look at the language $L = \{w \mid (|w|_\circ = \text{uneven}) \vee (|w|_\bullet = \text{uneven})\}$. These are all the words consisting of $\circ$'s and $\bullet$'s, where the ammount of $\circ$'s is uneven, or the ammount of $\bullet$'s is uneven, and not both. This language looks rather strange, but is a valid regular language, since we can construct a DFA that recognizes this language. This time we only give a state diagram. The state diagram of the DFA M can look as follows:



We see that it is fine to use arbitrary symbols as letters in the alphabet of a language. An example of a word that is in L is "$\bullet \bullet \circ \bullet \circ$". We also remark that it is possible for an automaton to have multiple accepting states, and that we can also give states arbitrary names, as M has as set states $Q = \{\kappa_0, \kappa_1, \kappa_2, \kappa_3\}$.

Example 2.1.7. The language $L = \{a^n b^n \mid n \in \mathbb{N}\}$ is a non-regular language. For this language a finite automaton can not count the exact number of a's, such that they correspond to the number of b's. Moreover, say we have an automaton that recognizes all words $a^n b^n$ up until a certain $n \in \mathbb{N}$, then the word $a^{n+1} b^{n+1}$ won't be accepted, which is in the language L.

2.2 Derivative languages

We introduce the notion of derivative language. If L is a language, a derivative language L_w is a set of words such that if we append a word from L_w to w we get a words that is in L itself.

Definition 2.2.1. Let L be a language over an alphabet Σ. The *w-derivative* of L is $L_w = \{u \mid wu \in L\}$. The set $\mathrm{Der}(L) = \{L_w \mid w \in \Sigma^*\}$ is the set of all derivatives of L.

Example 2.2.2. Take the language $L = \{a^*b^*c^* \mid a, b, c \in \Sigma\}$, where Σ is a finite alphabet. Let's look at the derivatives of L.

- $L_a = \{v \mid av \in \Sigma\} = \{a^*b^*c^* \mid a, b, c \in \Sigma\} = L$
- $L_b = \{v \mid bv \in \Sigma\} = \{b^*c^* \mid b, c \in \Sigma\} = L_{bb} = L_{ab}$
- $L_c = \{v \mid cv \in \Sigma\} = \{c^* \mid c \in \Sigma\} = L_{cc} = L_{ac} = L_{bc} = L_{abc}$
- $L_{ba} = \{v \mid bav \in \Sigma\} = \emptyset = L_{cb} = L_{ca} = L_{baa} = L_{bab} = L_{bac}$

We see that by adding more letters to the derivative languages, we get a derivative that we already have. For instance, we see that L_b is the same as L_{ab}, since the words v such that bv and abv are in L are exactly the same words. From this we may conclude that have found all derivatives, and thus we see that $\mathrm{Der}(L) = \{L_a, L_b, L_c, \emptyset\}$.

The following lemma will connect the definition of a derivative language to definition 2.1.3, which is the definition of state languages.

Lemma 2.2.3. *Let L be a language that is recognized by a DFA $M = (Q, \Sigma, \delta, q_0, F)$. Let $v \in \Sigma^*$, then $L_v = \mathcal{L}(\delta^*(q_0, v))$.*

Proof. We write out the definition of L_v and see that the equality holds:

$$\begin{aligned} L_v &= \{w \mid vw \in L\} \\ &= \{w \mid \delta^*(q_0, vw) \in F\} \\ &= \{w \mid \delta^*(\delta^*(q_0, v), w) \in F\} = \mathcal{L}(\delta^*(q_0, v)) \end{aligned}$$

□

Next we will see that we can also connect a word w being in a language L, to λ being in the derivative language L_w regarding that word w.

Lemma 2.2.4. *Let L be a language. Take $w \in \Sigma^*$, then $w \in L$ if and only if $\lambda \in L_w$.*

Proof. We prove this by induction on the word w.

- **Base case:** $w = \lambda$

$$\lambda \in L \iff \lambda \in L_\lambda = L$$

- **Induction step:** Assume that for word $w \in \Sigma^*$, $w \in L \iff \lambda \in L_w$ holds (IH). Note that this holds for an arbitrary language L. Take $x \in \Sigma$. Then for xw we have:

$$\begin{aligned} xw \in L &\iff w \in L_a = L \\ &\iff \lambda \in (L_x)_w \qquad \text{(Induction hypothesis)} \\ &\iff \lambda \in L_{xw} \end{aligned}$$

□

Now that we have introduced derivative languages, and we have seen some connections between languages and their derivatives, we can now prove a theorem that is somewhat more involved. The theorem gives us a new way of determining if a language L can be recognized by a DFA. This is by means of checking if $\mathrm{Der}(L)$ is a finite set. If this is the case, we can conclude that L is a regular language. Moreover, we will see that this relation is an equivalence, so the opposite will also hold.

Theorem 2.2.5. *Let L be a language over some alphabet Σ. Then L is recognized by a deterministic finite automaton if and only if $Der(L)$ is finite.*

Proof. Assume L is recognized by a deterministic finite automaton M. We see that:

$$\begin{aligned} \mathrm{Der}(L) &= \{L_w \mid w \in \Sigma^*\} \\ &= \{\mathcal{L}(\delta^*(q_0, w)) \mid w \in \Sigma^*\} \qquad \text{(lemma 2.2.3)} \\ &\subseteq \{\mathcal{L}(q) \mid q \in Q\} \end{aligned}$$

Since Q is finite, there are finitely many state languages $\mathcal{L}(q)$. Since $\mathrm{Der}(L)$ is a subset of these state languages, it has to be finite.
Now assume that $\mathrm{Der}(L)$ is finite. Consider the DFA $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \mathrm{Der}(L)$
- $\Sigma =$ the alphabet of L
- $q_0 = L$
- $F = \{L_i \mid \lambda \in L_i\}$
- $\delta(L_w, x) = L_{wx}$.

We now want to show that M recognizes the language L. This means that we want to show that $w \in L \iff \delta^*(q_0, w) \in F$. To do this we will first show that $\delta^*(L, w) = L_w$. This will be done by induction on the word w.

Base case: $w = \lambda$

$$\delta^*(L, \lambda) = L = L_\lambda$$

Induction step: Assume for $w \in \Sigma^*$ that $\delta^*(L, w) = L_w$ holds (IH). Take $x \in \Sigma$. Then for xw we have

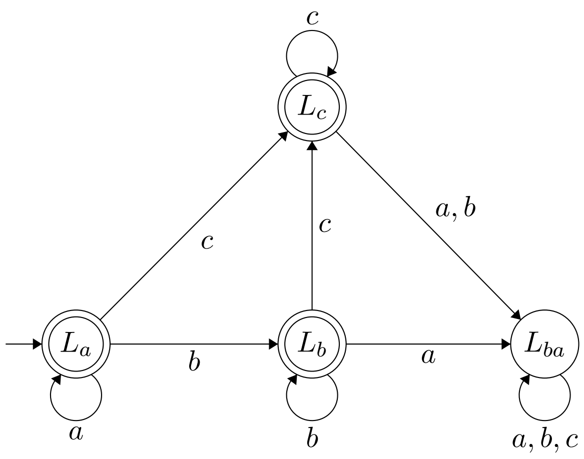
$$\begin{aligned}
\delta^*(L, xw) &= \delta^*(\delta(L, x), w) \\
&= \delta^*(L_x, w) \\
&= (L_x)_w && \text{(Induction hypothesis)} \\
&= L_{xw}
\end{aligned}$$

Now that we have seen that $\delta^*(L, w) = L_w$ we can derive the following:

$$\begin{aligned}
w \in L &\iff \lambda \in L_w && \text{(lemma 2.2.4)} \\
&\iff \lambda \in \delta^*(L, w) && \text{(Just shown)} \\
&\iff \lambda \in \delta^*(q_0, w) \\
&\iff \delta^*(q_0, w) \in F
\end{aligned}$$

□

Example 2.2.6. We will now use the construction from theorem 2.2.5 to create a deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ such that M recognizes L from example 2.2.2. We start by setting the set of states $Q = \mathrm{Der}(L)$. We have seen that $L_a = L$, hence initial state is L_a. Then we check for which derivative languages it holds that $\lambda \in L_w$. The state corresponding to those languages should final. We see that this is the case for L_a, L_b and L_c. Lastly we define the transition function δ so that $\delta(L_w, x) = L_v$ if $L_{wx} = L_v$. For example we see that $\delta(L_b, c) = L_c$ since $L_{bc} = L_c$. The state diagram of the resulting automaton then looks like:



2.3 Homomorphisms and substitutions

We are interested in transforming languages, so that we can investigate closure properties. Take languages L_1 and L_2. Classical ways of transforming languages are for example taking the complement $\overline{L_1}$, the union $L_1 \cup L_2$ or the intersection $L_1 \cap L_2$. In the context of this thesis we are interested in homomorphisms and substitutions between languages. We will define what these are, and with these definitions we will show that regular languages are closed under substitutions.

Definition 2.3.1. Let Σ and Δ be two alphabets. Take a function $h : \Sigma \to \Delta^*$. We extend this function to $h^\sharp : \Sigma^* \to \Delta^*$ such that the following property holds:

$$\forall x, y \in \Sigma^* \; [h^\sharp(xy) = h^\sharp(x)h^\sharp(y)]$$

where $h^\sharp(x) = h(x)$ for $x \in \Sigma$. A function $h^\sharp$ with this property is called a *homomorphism.* Since $h^\sharp$ is derived from h, we also say that h is a homomorphism. We see that $h^\sharp$ is now defined for words in Σ^*. Using the property, we see that following holds for $w \in \Sigma^*$:

$$h^\sharp(w) = h^\sharp(a_1a_2\ldots a_n) = h^\sharp(a_1)h^\sharp(a_2)\ldots h^\sharp(a_n) = h(a_1)h(a_2)\ldots h(a_n).$$

Definition 2.3.2. Let Σ and Δ be two alphabets. Take a function $s : \Sigma \to \mathcal{P}(\Delta^*)$. We extend this function to $s^\sharp : \Sigma^* \to \mathcal{P}(\Delta^*)$ such that the following property holds:

$$\forall x, y \in \Sigma^* \; [s^\sharp(xy) = s^\sharp(x)s^\sharp(y)]$$

where $s^\sharp(x) = s(x)$ for $x \in \Sigma$. If $s^\sharp(x)$ and $s^\sharp(y)$ are both sets, $s^\sharp(x)s^\sharp(y)$ contains all words vw such that $v \in s^\sharp(x)$ and $w \in s^\sharp(y)$. A function $s^\sharp$ with the above property is called a *substitution.* Since $s^\sharp$ is derived from s, we also say that s is a substitution. We see that $s^\sharp$ is now defined for words in Σ^*. Using the property, we see that following holds for $w \in \Sigma^*$:

$$s^\sharp(w) = s^\sharp(a_1a_2\ldots a_n) = s^\sharp(a_1)s^\sharp(a_2)\ldots s^\sharp(a_n) = s(a_1)s(a_2)\ldots s(a_n).$$

We note that a substitution $s : \Sigma \to \mathcal{P}(\Delta^*)$ sends a letter $x \in \Sigma$ to a language $L \subseteq \Delta^*$. Take a homomorphism $h : \Sigma \to \Delta^*$. Then the image $h(x)$ of a letter $x \in \Sigma$ can be see as a singleton set, which is a subset of $\mathcal{P}(\Delta^*)$, and thus a language. This means we could also define $h : \Sigma \to \mathcal{P}(\Delta^*)$, hence every homomorphism is also a substitution. This means that if we prove a property for substitutions, then we also get that this property holds for homomorphisms. We now give some examples of homomorphisms and substitutions, and we show what the result is if we apply them to words.

Example 2.3.3. Let $\Sigma = \{a, b, c, d, \ldots, x, y, z\}$, the Latin alphabet. Then we define homorphism $h : \Sigma \to \Sigma^*$ as $h(a) = b,\ h(b) = c,\ \ldots h(y) = z,\ h(z) = a$. This homomorphism shifts every letter one place, in the alphabet Σ. If we look at the extension $h^\sharp$ which is defined for words in Σ^* we for instance get: $h^\sharp(nominal) = opnjobm$.

Example 2.3.4. Let $\Sigma = \{a, b, c, d, \ldots, x, y, z\}$, the Latin alphabet. Then we define homorphism $h : \Sigma \to \Sigma^*$ as $h(\alpha) = \alpha\alpha$ for $\alpha \in \Sigma$. This homomorphism doubles every letter in a word. If we take the extension $h^\sharp$ we for instance get: $h^\sharp(automata) = aauuttoommaattaa$.

Example 2.3.5. Let $\Sigma = \{a, b, c, d, \ldots, x, y, z\}$, the Latin alphabet. Then we define substitution $s : \Sigma \to \mathcal{P}(\Sigma^*)$ as $s(\alpha) = \Sigma \setminus \{\alpha\}$, for $\alpha \in \Sigma$. This substitution sends a letter to the complement of that letter, regarding the alphabet Σ. If we take the extension $s^\sharp$ we for instance get:

$$\begin{aligned} s^\sharp(deterministic) &= s^\sharp(d)s^\sharp(e)\ldots s^\sharp(i)s^\sharp(c) \\ &= (\Sigma \setminus \{d\})(\Sigma \setminus \{e\})\ldots(\Sigma \setminus \{i\})(\Sigma \setminus \{c\}) \\ &= \{w \mid w \text{ has no letters in common with} \\ &\qquad \text{the word } deterministic \text{ and } |w| = 13\} \end{aligned}$$

Because regular languages can be recognized by DFA's, we would like to investigate if regularity is preserved under substitutions. This closure would mean, that for a regular language, it's image under a substitution is also a regular language. The image then can also be recognized by a DFA.

Having this closure might come in handy when investigating if a language is regular. One could define a substitution, and then look at the image of the language to be investigated. If the image is regular, the original language also has to be regular.

Definition 2.3.6. Let Σ and Δ be finite alphabets. Let $L \subseteq \Sigma^*$. Let $s^\sharp : \Sigma^* \to \mathcal{P}(\Delta^*)$ be a substitution derived from s. Then we define the application of the substitution on a language as $s^\sharp(L) = \bigcup_{w \in L} s^\sharp(w)$

The following lemma and theorem can be found in [7]. Loc. cit. the lemma and the theorem are put together in a single theorem. For clarity we have split this theorem into a lemma and a theorem. Also in the work of Shallit the parts that use induction could be more extensive, so we tried to write out these part including base cases and induction hypotheses.

Lemma 2.3.7. *Let Σ be a finite alphabet. Let $L_1, L_2 \subseteq \Sigma^*$. Let $s^\sharp : \Sigma^* \to \mathcal{P}(\Sigma^*)$ be an extension of a substitution s. Then the following hold:*

$$\begin{array}{lrcl} (i) & s^\sharp(L_1 \cup L_2) & = & s^\sharp(L_1) \cup s^\sharp(L_2) \\ (ii) & s^\sharp(L_1 L_2) & = & s^\sharp(L_1)s^\sharp(L_2) \\ (iii) & s^\sharp(L^*) & = & s^\sharp(L)^* \end{array}$$

Proof. Part (i): We will prove the more general case, using an arbitrary index set $\mathcal{I}$ with $L_i \subseteq \Sigma^*$, that states that $s^\sharp(\bigcup_{i\in\mathcal{I}} L_i) = \bigcup_{i\in\mathcal{I}} s^\sharp(L_i)$. This is the result we will use in the theorem to come.

$$\begin{aligned} s^\sharp(\bigcup_{i\in\mathcal{I}} L_i) &= \bigcup_{w\in\bigcup_{i\in\mathcal{I}} L_i} s^\sharp(w) \\ &= \bigcup_{i\in\mathcal{I}} \bigcup_{w\in L_i} s^\sharp(w) \\ &= \bigcup_{i\in\mathcal{I}} s^\sharp(L_i) \end{aligned}$$

Part (ii):

$$\begin{aligned} s^\sharp(L_1L_2) &= \bigcup_{w\in L_1L_2} s^\sharp(w) \\ &= \bigcup_{w_1\in L_1, w_2\in L_2} s^\sharp(w_1w_2) \\ &= \bigcup_{w_1\in L_1, w_2\in L_2} s^\sharp(w_1)s^\sharp(w_2) \\ &= \left(\bigcup_{w_1\in L_1} s^\sharp(w_1)\right)\left(\bigcup_{w_2\in L_2} s^\sharp(w_2)\right) \\ &= s^\sharp(L_1)s^\sharp(L_2) \end{aligned}$$

Part (iii): This part we will proof by induction on $n \in \mathbb{N}$.

Base case:

$$s^\sharp(L^0) = s^\sharp(\{\lambda\}) = \{\lambda\} = s^\sharp(L)^0$$

Inductive step: Assume that $s^\sharp(L^n) = s^\sharp(L)^n$ holds (IH). We show that $s^\sharp(L^{n+1}) = s^\sharp(L)^{n+1}$.

$$\begin{aligned} s^\sharp(L^{n+1}) &= s^\sharp(L^nL) & \\ &= s^\sharp(L^n)s^\sharp(L) & \text{(Part } (ii)\text{)} \\ &= s^\sharp(L)^n s^\sharp(L) & \text{(Induction hypothesis)} \\ &= s^\sharp(L)^{n+1} & \end{aligned}$$

Now

$$\begin{aligned} s^\sharp(L^*) &= s^\sharp(\textstyle\bigcup_{i\geq 0} L^i) & \\ &= \textstyle\bigcup_{i\geq 0} s^\sharp(L^i) & \text{(Part } (i)\text{)} \\ &= s^\sharp(L)^i & \text{(Just proven by induction)} \\ &= s^\sharp(L)^* & \end{aligned}$$

□

Theorem 2.3.8. *Regular languages are closed under substitutions.*

Proof. Let L be a regular language, and let $s^\sharp : \Sigma^* \to \mathcal{P}(\Delta^*)$ be the extension of substitution s. Since L is a regular language, we know that there exists a regular expressions r so that $\mathcal{L}(r) = L$. We will prove the theorem by structural induction on the regular expression r.

Base case $r = 0$:
$\mathcal{L}(r) = \emptyset$ and $s^\sharp(\emptyset) = \emptyset$, and thus $s^\sharp(\emptyset)$ is regular.

Base case: $r = 1$:
$\mathcal{L}(r) = \{\lambda\}$ and $s^\sharp(\lambda) = \{\lambda\}$, and thus $s^\sharp(\lambda)$ is regular.

Base case: $r = a$:
$\mathcal{L}(r) = a$. Since $s^\sharp$ sends a letter to a regular language by definition, $s^\sharp(a)$ is regular.

Assume $s^\sharp(\mathcal{L}(r_1))$ and $s^\sharp(\mathcal{L}(r_2))$ are regular (IH).

Inductive step: $r = r_1 + r_2$:

$$\begin{aligned} s^\sharp(\mathcal{L}(r)) &= s^\sharp(\mathcal{L}(r_1 + r_2)) \\ &= s^\sharp(\mathcal{L}(r_1) \cup \mathcal{L}(r_2)) \\ &= s^\sharp(\mathcal{L}(r_1)) \cup s^\sharp(\mathcal{L}(r_2)) \qquad \text{(lemma 2.3.7}(i)\text{)} \end{aligned}$$

Now with use of the induction hypothesis we see that $s^\sharp(\mathcal{L}(r))$ is regular, since the union of two regular languages is again a regular language.

Inductive step: $r = r_1 r_2$:

$$\begin{aligned} s^\sharp(\mathcal{L}(r)) &= s^\sharp(\mathcal{L}(r_1 r_2)) \\ &= s^\sharp(\mathcal{L}(r_1)\mathcal{L}(r_2)) \\ &= s^\sharp(\mathcal{L}(r_1))s^\sharp(\mathcal{L}(r_2)) \qquad \text{(lemma 2.3.7}(ii)\text{)} \end{aligned}$$

Now with use of the induction hypothesis we see that $s^\sharp(\mathcal{L}(r))$ is regular, since the concatenation of two regular languages is again a regular language.

Inductive step: $r = r_1^*$:

$$\begin{aligned} s^\sharp(\mathcal{L}(r)) &= s^\sharp(\mathcal{L}(r_1^*)) \\ &= s^\sharp(\mathcal{L}(r_1)^*) \\ &= s^\sharp(\mathcal{L}(r_1))^* \qquad \text{(lemma 2.3.7}(iii)\text{)} \end{aligned}$$

Now with use of the induction hypothesis we see that $s^\sharp(\mathcal{L}(r))$ is regular, since the Kleene star applied to a regular language, results in a regular language. This completes the proof by structural induction on regular expressions, and thus we may conlude that regular languages are closed under substitutions. □

Chapter 3

Groups and nominal sets

3.1 Group theory and G-sets

In mathematics and abstract algebra, the field of group theory studies so-called *groups.* Groups are sets, equipped with a binary operator, that satisfy axioms. A lot of other structures in abstract algebras, such as rings, fields and vector spaces, can be interpreted as a group, with some extra axioms and operators. A classic example of a group structure can be found in a Rubik's cube, where turning a side can be seen as a group element [3].

The main definitions are taken from the lecture notes for the course "Groepentheorie" at Radboud University [4].

Definition 3.1.1. A *group* $\langle G, \circ\rangle$ is a set G together with an operation $\circ : G \times G \to G$ such that $\circ$ satisfies the following axioms:

$$\begin{array}{llll}
(G1) & \textit{Associativity}: & \forall \pi, \sigma, \varphi \in G & [(\pi \circ \sigma) \circ \varphi = \pi \circ (\sigma \circ \varphi)] \\
(G2) & \textit{Closure}: & \forall \pi, \sigma \in G & [\pi \circ \sigma \in G] \\
(G3) & \textit{Identity}: & \exists \varepsilon \in G \; \forall \pi \in G & [\varepsilon \circ \pi = \pi = \pi \circ \varepsilon] \\
(G4) & \textit{Inverse}: & \forall \pi \in G \; \exists \pi^{-1} \in G & [\pi \circ \pi^{-1} = \varepsilon = \pi^{-1} \circ \pi]
\end{array}$$

Proposition 3.1.2. *The identity element of a group is unique.*

Proof. Let G be a group. Let ε and δ both be identity elements of G. Then we see with $(G3)$ that: $\varepsilon = \varepsilon \circ \delta = \delta$. And thus ε and δ are the same element. □

Proposition 3.1.3. *The inverse element for a group element is unique.*

Proof. Let G be a group. Let $\pi \in G$. Then assume that both σ and φ are inverse elements for π. Then with help of the group axioms, we get:

$$\begin{array}{rcll} \varphi & = & \varphi \circ \varepsilon & (G3) \\ & = & \varphi \circ (\pi \circ \sigma) & (G4) \\ & = & (\varphi \circ \pi) \circ \sigma & (G1) \\ & = & \varepsilon \circ \sigma & (G4) \\ & = & \sigma & (G3) \end{array}$$

□

Example 3.1.4. A simple example of group is for instance $\langle \mathbb{Z}, + \rangle$. One can prove that $+$ is an associative operator. If we add two elements of $\mathbb{Z}$, we again get an element of $\mathbb{Z}$. We have identity 0 and for every $x \in \mathbb{Z}$ there exists $-x \in \mathbb{Z}$ so that $x + (-x) = 0$.

An important example of a group is *the permutation group* [4] (chapter 4). For a set X the permutation group of X is notated as $\mathbf{Perm}(X)$. The permutation group of X consists of all possible permutations of X. A *permutation* is a mapping σ that maps all elements of X to another elements of X. The reason these permutation groups are important, is because we will use them when defining nominal sets.

Example 3.1.5. Take $X = \{1, 2, 3, 4\}$, then an element of the permutation group of X could be $\sigma = (2134)$. This element maps 2 to 1, 1 to 3, 3 to 4 and 4 to 2. This can also be written as $\sigma(2) = 1$, $\sigma(1) = 3$, $\sigma(3) = 4$ and $\sigma(4) = 2$. The permutation group $\mathbf{Perm}(X)$ consists of all possible permutations σ one can make for X. Do note that some elements might look different but are the same, i.e. (1234) and (3412) are the same element, only shifted around.

Most often, elements of a group can be used to transform elements of a set. Lets again look at the Rubik's cube example. A turn of a side can be seen as group element. We can apply this turn-element to a configuration of a Rubik's cube, to get a different configuration. The group element has then acted on the set of all configurations.

Definition 3.1.6. Let X be an ordinary set, and let G be a group. A *right group action* is a function $\cdot : X \times G \to X$ such that the following properties are satisfied:

$$\begin{array}{ll} (A1) & \forall x \in X\ \forall \pi, \sigma \in G[x \cdot (\pi \circ \sigma) = (x \cdot \pi) \cdot \sigma] \\ (A2) & \forall x \in X[x \cdot \varepsilon = x] \end{array}$$

The set X is called a *G-set* and we say that G *acts* on X, or that G is *acting* on X.

Example 3.1.7. Take the set $X = \{a, b, c, \ldots, x, y, z\}$. We will look at a subset of $\mathbf{Perm}(X)$. Take $G = \{\pi_i \mid 0 \leq i \leq 25\}$ where π_i shifts a letter i places along the set X. The group action $\cdot : X \times G \to X$ would then work as follows: $\pi_0 \cdot a = a,\ \pi_1 \cdot a = b,\ \pi_2 \cdot a = c,\ \ldots,\ \pi_{24} \cdot a = y,\ \pi_{25} \cdot a = z$ and $\pi_0 \cdot b = b,\ \pi_1 \cdot b = c,\ \pi_2 \cdot b = d,\ \ldots,\ \pi_{24} \cdot b = z,\ \pi_{25} \cdot b = a$, and so on for every element of X. We can see that G is a group. For $\pi_i, \pi_j \in G$ we have $\pi_i \circ \pi_j = \pi_{i+j \bmod 25} \in G$. The neutral element of G is π_0. Associativity also holds because $+$ is associative, and we see that unique inverses exist since $\forall i \in \{0, \ldots, 25\} \exists i \in \{0, \ldots, 25\}[i + j = 0 \bmod 25]$. We can also see that $\cdot$ is a group action, since π_0 indeed acts as an identity, and $\forall x \in X \forall \pi_i, \pi_j \in G[x \cdot (\pi_i \circ \pi_j) = x \cdot \pi_{i+j \bmod 25} = (x \cdot \pi_i) \cdot \pi_j]$

3.2 Orbits and equivariance

From now on, if we mention *"a G-set X"*, we implicitly assume that there is a group G that acts on X. This means that for a G-set we can talk about $\pi \in G$, or about a group-action π in general, without explicitly introducing the group G .

Up until now we have only seen an example of a G-set that is finite. In general, a G-set can also be infinite. For such a G-set, we would like to introduce some more structure. The structure we are introducing are so-called *orbits.*

Definition 3.2.1. Let X be a G-set. Take $x \in X$. The *orbit* of x is

$$x \cdot G = \{x \cdot \pi \mid \pi \in G\} \subseteq X.$$

The G-set X is called *orbit-finite* if it has a finite amount of orbits, i.e. the set

$$X \cdot G = \{x \cdot G \mid x \in X\}$$

is finite.

Example 3.2.2. The G-set X from example 3.1.7 is orbit finite. If we look at the orbit of the letter $a \in X$ we can see this is actually the whole set X:

$$a \cdot G = \{a \cdot \pi_n \mid \pi_n \in G\} = \{a \cdot \pi_0, a \cdot \pi_1, \ldots, a \cdot \pi_{25}\} = \{a, b, \ldots, z\} = X.$$

Moreover, the orbit of any letter is the whole set. This means that all letters are in the same orbit, or in other words, that X is a single-orbit set.

In the next example we will look at a product of two G-set, which is a G-set itself. If we have G-set X, then $X \times X$ is also a G-set, because the group axioms are preserved under Cartesian product. Take an element (x, y) of $X \times X$. Then a group element π from the set G acts on (x, y) as follows:

$$(x, y) \cdot \pi = (x \cdot \pi, y \cdot \pi).$$

Example 3.2.3. We again take $X = \{a, b, c, \ldots, x, y, z\}$. Then, we look at the set $X \times X = \{(\alpha_1, \alpha_2) \mid \alpha_1, \alpha_2 \in X\}$. The for the group acting on $X \times X$ we take $G = \mathbf{Perm}(X)$. Now $X \times X$ is a G-set and has two orbits. One is the orbit of the element (a, a). Every element in this orbit has the property that both elements are equal. Then the second orbit belongs to the and element (a, b), where the two elements in the tuple are not the same. This orbit consists of all the tuples in $X \times X$, for which the two elements are not the same.

We now introduce the notion of equivariance. A G-set is equivariant if the set is unchanged, after applying a group element on the set. Later in this thesis we will define so-called *nominal substitutions*, which we want to be equivariant. This will imply that for a substitution s it will hold that $s(x) \cdot \pi = s(x \cdot \pi)$.

Definition 3.2.4. Let X be a G-set. A set $Y \subseteq X$ is called *equivarant* if $\forall \pi \in G\ [Y \cdot \pi = Y]$.

Proposition 3.2.5. *Let X be a G-set. Let $Y \subseteq X$ be an equivariant subset of X. Then Y is a union of orbits.*

Proof. Assume that Y is not a union of orbits. Then there has to be an element $y \in Y$ such that there is a $\pi \in G$ for which $y \cdot \pi \notin Y$, because otherwise the whole orbit $y \cdot G$ would be in Y. But this gives a contradiction, since then Y is not an equivariant set. $\square$

Remark 3.2.6. Let X be a G-set. Let $Y \subseteq X$ be an orbit of X. Then for $y \in X$ we have $Y = \{y \cdot \pi \mid \pi \in G\} = y \cdot G$, and we see that Y is an equivariant set.

Example 3.2.7. The G-set X from example 3.1.7 is equivariant. We need X to be a subset of a G-set, so we take $X \subseteq X$. Then we see that $\forall \alpha \in X\ \forall \pi \in G[\alpha \cdot \pi \in X]$, and thus $\forall \pi \in G[X \cdot \pi = X]$. We can also see this since X itself is an orbit, and thus equivariant by remark 3.2.6.

Since functions, can also be regarded as sets of tuples, we can define what an equivariant function is. From this we will see what it means for a substitution to be equivariant, since it is a function.

Definition 3.2.8. Let X and Y be two G-sets. A function $f : X \to Y$ is called an *equivariant function* if for any $\pi \in G$ it holds that $f(x \cdot \pi) = f(x) = f(x) \cdot \pi$.

Example 3.2.9. Take $X = \{a, b, c, \ldots, x, y, z\}$ and take the group G as in example 3.1.7. An example of an equivariant function $i : X \to X$, the identity mapping. We see that for an arbitrary element $\alpha \in X$,

$$f(\alpha) \cdot \pi = \alpha \cdot \pi = f(\alpha \cdot \pi).$$

Example 3.2.10. Take $X = \{a, b, c, \ldots, x, y, z\}$ and take the group G as in example 3.1.7. An example of a function that is not equivariant is $g(\alpha) =$ "the next letter in the alphabet". Take $\pi \in G$ such that π sends all letters to the letter a. Then

$$g(b \cdot \pi) = g(a) = b \neq a = c \cdot \pi = g(b) \cdot \pi.$$

3.3 Nominal sets

In the previous section we assumed that when we say *"a G-set X"*, we implicitly assume that there is a group G that acts on X. From now on we will make another assumption, namely that the group G is $\mathbf{Perm}(\mathbb{A})$, the permutation group of the so-called *atoms.* The set of atoms $\mathbb{A}$ is the infinite set of minimal elements. We will now introduce *nominal sets.* These sets are useful when modeling computations over the atoms. To be able to define nominal sets we first have to explain what it means for an element to be *supported.*

Definition 3.3.1. Let X be a G-set. A set $C \subseteq \mathbb{A}$ *supports* an element $x \in X$ if $\forall c \in C[c \cdot \pi = c] \implies x \cdot \pi = x$.

First we remark that the set C does not need to be unique. We illustrate this by the following example:

Example 3.3.2. Take the set of atoms $\mathbb{A}$. Then take an element $a \in \mathbb{A}$. Then a is supported by the set $\{a\}$. But it is also supported by the set $\{a, b\}$. In fact a is supported by any set C for which $a \in C$.

Now that we know what it means for an element to be supported, we can define what a nominal set is.

Definition 3.3.3. Let X be a G-set. X is *nominal* is every $x \in X$ is supported by some finite support C.

Example 3.3.4. Take the set of atoms $\mathbb{A}$. Then $\mathbb{A}$ itself is a nominal set, since we have seen that every $a \in \mathbb{A}$ is supported by the set $\{a\}$.

Proposition 3.3.5. *Let X be a nominal set. Then X^* is a nominal set.*

Proof. If we look at X^* as:

$$X^* = \bigcup_{n \geq 0} X^n$$

then an arbitrary element of this set is $(x_1, \ldots, x_n)$ and this element has finite support $\{x_1, \ldots, x_n\}$, so the set X^* is nominal. $\square$

Example 3.3.6. Take the set of atoms $\mathbb{A}$. Then power set of $\mathbb{A}$, $\mathcal{P}(\mathbb{A})$ is not a nominal set. Since $\mathbb{A}$ is an infinite set there are sets $X \in \mathcal{P}(\mathbb{A})$ that are not finite, and not co-finite, the latter meaning that they don't have a finite complement. These sets have no finite support, hence $\mathcal{P}(\mathbb{A})$ is not a nominal set. Since $\mathcal{P}(\mathbb{A})$ is not nominal, $\mathcal{P}(\mathbb{A}^*)$ is also not nominal, with similar reasoning.

The next two lemma's of this subsection connect equivariant sets and functions to things we have just seen, i.e. nominal sets and support. These lemma's play a key role in a theorem to come, where we want to prove properties about nominal substitutions.

Lemma 3.3.7. *An equivariant subset of a nominal set is a nominal set.*

Proof. Let X be a nominal set. And let $Y \subseteq X$ be an equivariant subset. This means that we have

$$\forall \pi \in G \; [Y \cdot \pi = Y].$$

Since $Y \subseteq X$, we know that for every $y \in Y$, that it is also an element of X, and since X is nominal we know that y then has a finite support C. This means that

$$\forall \pi \in G \; [\forall c \in C \; [c \cdot \pi = c] \implies y \cdot \pi = y].$$

So Y is a nominal set. $\square$

Lemma 3.3.8. *An equivariant function preserves support.*

Proof. Let X and Y be two G-sets. Let $f : X \to Y$ be an equivariant function. We need to prove for an element $x \in X$, that if x is supported by a support C, then $f(x)$ is also supported by C. Let C be a support for $x \in X$. This means that

$$\forall \pi \in G \; [\forall c \in C \; [c \cdot \pi = c] \implies x \cdot \pi = x.$$

We want to show that C is also a support for $f(x)$. This means that

$$\forall \pi \in G \; [\forall c \in C \; [c \cdot \pi = c] \implies f(x) \cdot \pi = f(x).$$

We now use that f is equivariant, and the fact that $x \cdot \pi = x$, and see that

$$f(x) \cdot \pi = f(x \cdot \pi) = f(x).$$

We thus conclude that f preserves finite support. $\square$

The last definition in this section is the finitely supported power set. The name does a pretty good job of explaining how it is defined. It is a last piece of theory we need, to be able to give a nice definition of nominal substitutions in the next section. This is because the domain and the range have to be finitely supported power sets to make substitutions work.

Definition 3.3.9. Let X be a set. The *finitely supported power set of* X is

$$\mathcal{P}_{fs}(X) = \{Y \subseteq X \mid Y \text{ has finite support } C\}$$

Remark 3.3.10. Let X be a set. Then $\mathcal{P}_{fs}(X)$ is a nominal set, since all of its elements are finitely supported by definition.

Chapter 4

Nominal automata theory

In the final chapter of this thesis we will investigate the behaviour of the just defined nominal languages under nominal substitutions. These nominal substitutions we will define in this chapter. Another thing we will look into in this chapter are nominal deterministic orbit finite automata. This flavour of automata is able to recognize nominal languages.

4.1 Nominal languages

Definition 4.1.1. Let X be an orbit finite nominal G-set. A *nominal language* is an equivariant subset $L \subseteq X^*$. We say that X is the alphabet of L, since words in L consist of elements from X.

Example 4.1.2. Take the language $L = \{awa \mid a \in \mathbb{A}, w \in \mathbb{A}^*\}$, where the atoms $\mathbb{A}$ is the alphabet. We see that $L \subseteq \mathbb{A}^*$. For L to be a nominal set, we want L to be an equivariant set. This means the $\forall \pi \in G[L \cdot \pi = L]$. If we look at $(awa) \cdot \pi = (a \cdot \pi)(a \cdot \pi)(a \cdot \pi)$, we see that if we take an arbitrary group element $\pi \in G$, and an arbitrary word $awa \in L$, that $(awa) \cdot \pi \in L$, because $a \cdot \pi \in \mathbb{A}$, and $w \cdot \pi \in \mathbb{A}^*$. This means that L is equivariant, and thus we can conclude that L is an nominal language.

In the definition of a nominal language L, it might seem strange we do not require L to be a nominal set, a priori. If we take a closer look we see that it actually follows the equivariance that the language L is indeed a nominal set.

Proposition 4.1.3. *Any nominal language is a nominal set.*

Proof. Let L be a nominal language. This means that there is a nominal set X such that $L \subseteq X^*$, and that L is a equivariant set. By definition we know that X^* is also a nominal set. Then from lemma 3.3.7 we get that an equivariant subset of a nominal set is a nominal set itself, hence L is a nominal set. $\square$

The above lemma shows us that we may apply theory that we have developed for nominal sets, to nominal languages. This is because since every nominal language is a nominal set.

For regular languages we have seen what a derivative language is. For nominal languages this definition is basically the same. For good measure we again give the definition, with slight alterations for the nominal setting.

Definition 4.1.4. Let X be an orbit finite nominal G-set. Let $L \subseteq X^*$ be a nominal set. Let $w \in X^*$. The *nominal w-derivative* of L is $L_w = \{v \mid wv \in L\}$. The set $\mathrm{Der}(L) = \{L_w \mid w \in X^*\}$ is the set of all nominal derivatives that we can deduce from L.

We will not always explicitly say that a set L_w is a nominal w-derivative, but just a w-derivative. From the setting one can deduct if a w-derivative is nominal or not. All the words in L_w are now also an element of X^*, which means that group elements $\pi \in G$ now also can act on $u \in L_w$.

Definition 4.1.5. Let X be a nominal orbit finite G-set. Let $L \subseteq X^*$ be a nominal language. Then we define $L_w \cdot \pi = \{u \mid wu \in L\} \cdot \pi$

Lemma 4.1.6. *If we have a nominal language L, then for $L_w \in Der(L)$ and $\pi \in G$:*

$$L_w \cdot \pi = L_{w\cdot\pi}.$$

Proof.

$$\begin{aligned}
L_w \cdot \pi &= \{u \mid wu \in L\} \cdot \pi \\
&= \{u \cdot \pi \mid wu \cdot \pi \in L \cdot \pi\} \\
&= \{u \cdot \pi \mid (w \cdot \pi)(u \cdot \pi) \in L\} && (L \text{ is equivariant}) \\
&= \{v \mid (w \cdot \pi)v \in L\} && (u \cdot \pi = v \in L) \\
&= L_{w\cdot\pi}
\end{aligned}$$

□

Example 4.1.7. Take the language L from example 4.1.2. If we then fix an element $a \in \mathbb{A}$, examples of derivatives are $L_a = \{wa \mid, w \in \mathbb{A}^*\}$, $L_{aba} = \{\lambda\}$. By the previous lemma we can also see group actions on these derivative languages. Say we have a group action π that changes the fixed element a into $b \in \mathbb{A}$ and vice versa. Then applying the group action the the derivative examples gives $L_a \cdot \pi = L_{a\cdot\pi} = L_b = \{wb \mid w \in \mathbb{A}^*\}$, and $L_{aba} \cdot \pi = L_{(aba)\cdot\pi} = L_{bab} = \{\lambda\}$. We see that in some cases that the derivative language also changes according to the group action, and in some cases it does not. This gives some reason to investigate the behaviour of the set $\mathrm{Der}(L)$.

We will now prove some properties about the set $\mathrm{Der}(L)$. In the next subsection we will construct automata where we use derivative languages as states. We want that the set of states is a nominal set. This is what we will now prove for $\mathrm{Der}(L)$.

Lemma 4.1.8. *Let X be a nominal orbit finite G-set. Let $L \subseteq X^*$ be a nominal language. Then Der(L) $\subseteq \mathcal{P}_{fs}(X^*)$.*

Proof. Let L be a nominal language. We have seen that for a nominal language L, by lemma 4.1.6 we have $L_w \cdot \pi = L_{w \cdot \pi}$. Take $L_w \in \mathrm{Der}(L)$, and write $w = x_1 x_2 \ldots x_n$, where $x_i \in X$. Then we look at $C = \{x_1, x_2, \ldots, x_n\}$. Then we see that because C support every letter of w, it is a support for w. This means that

$$\forall \pi \in G[\forall c \in C[c \cdot \pi = c] \implies w \cdot \pi = w].$$

Now we can derive that for an arbitrary $L_w \in \mathrm{Der}(L)$ we have

$$\forall \pi \in G[\forall c \in C[c \cdot \pi = c] \implies L_w \cdot \pi = L_{w \cdot \pi} = L_w].$$

Thus we see that every L_w is finitely supported by the C we proposed, hence $\mathrm{Der}(L) \subseteq \mathcal{P}_{fs}(X^*)$. □

Lemma 4.1.9. *Let X be a orbit finite nominal G-set. Let $L \subseteq X^*$ be a nominal language, then Der(L) is a nominal set.*

Proof. From lemma 4.1.6 we know that $\mathrm{Der}(L)$ is an equivariant set. Then from lemma 4.1.8 we know that $\mathrm{Der}(L) \subseteq \mathcal{P}_{fs}(X^*)$, and from remark 3.3.10 we know that $\mathcal{P}_{fs}(X^*)$ is a nominal set. We may then conclude from lemma 3.3.7, that $\mathrm{Der}(L)$ is a nominal set, because it is an equivariant subset of a nominal set. □

Example 4.1.10. Take the language L from example 4.1.2. We have seen that L is a nominal language. So from the above lemma it has to hold that $\mathrm{Der}(L)$ is a nominal set. This also means that $\mathrm{Der}(L)$ is orbit finite. These orbits are represented by the derivatives L_λ, L_a and L_{aa}.

4.2 Nominal deterministic finite automata

We will now define a class of automata that can compute over nominal sets. These automata can not recognize nominal languages L in general, but only if the set $\mathrm{Der}(L)$ is orbit finite. We will achieve this result by proving that given an nominal DFA that recognized a language L, the set $\mathrm{Der}(L)$ is orbit finite. Then conversely, asssuming that for a language L the set $\mathrm{Der}(L)$ is orbit finite, that we can construct a proper nominal DFA that recognizes L. Let's first start by defining what a nominal DFA is.

Definition 4.2.1. A *nominal deterministic finite automaton* or *nominal DFA* is an automaton $M = (Q, \Sigma, \delta, q_0, F)$ where the set of states Q is an orbit finite nominal set, the alphabet Σ is an orbit finite nominal set, the transition function δ is an equivariant function, and the initial state q_0 and the set of final states F is an equivariant subset of Q.

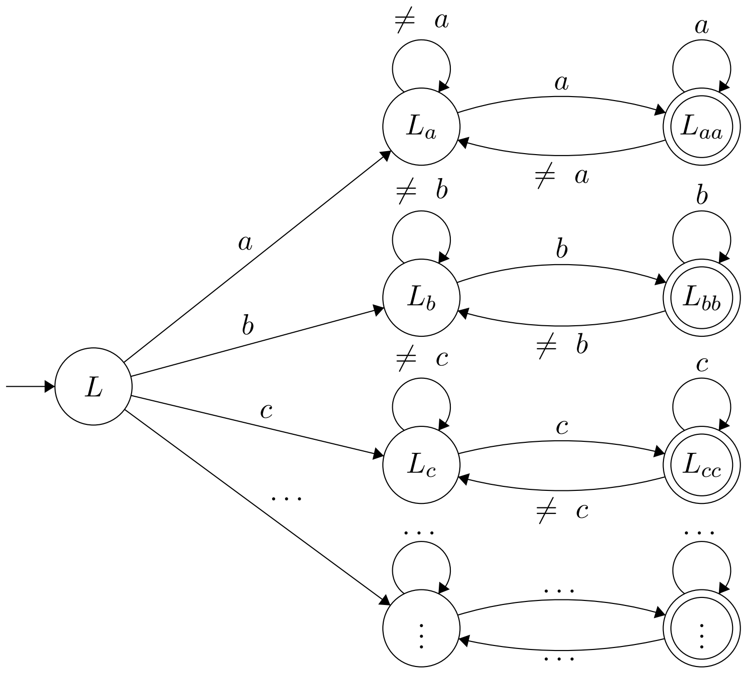
Definition 4.2.2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a nominal DFA. For every state $q \in Q$ we define $\mathcal{L}(q) = \{w \mid \delta^*(q, w) \in F\}$ as the *state language of* q.

We introduce a construction called the *syntactical automaton*. This is the construction we will use to prove one side of the earlier suggested theorem. We will show that if we assume that the set $\mathrm{Der}(L)$ is orbit finite, that this construction gives us a proper nominal DFA as defined above. Note that the syntactical automaton is similar to the automaton that was constructed in theorem 2.2.5. We chose to explicitly define it here, because using this construction is more involved in the nominal setting.

Definition 4.2.3. Let X be an orbit finite nominal G-set. Let $L \subseteq X^*$ be an nominal language. We define the *syntactical automaton* M_L as:

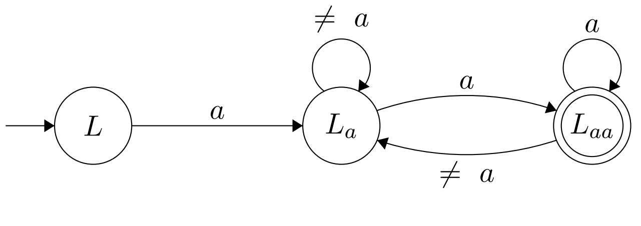
$$\begin{aligned}
Q &= \mathrm{Der}(L)\\
\Sigma &= X\\
\delta(L_w, x) &= L_{wx}\\
q_0 &= L\\
F &= \{L_w \mid w \in L\}
\end{aligned}$$

Example 4.2.4. For the language $L = \{awa \mid a \in A, w \in \mathbb{A}^*\}$, as was described in example 4.1.2, we can now also construct the syntactical automata. We will first give the state diagram of this automata, in such a way that all the state are displayed. This is one way of displaying M_L:



The first state is the initial state, which by definition is L. Then for each element $x \in \mathbb{A}$ the there is a transition from L to L_x. There are infinitely many states, since $\mathbb{A}$ is infinite. This is represented but the dots at the bottom, as we can not list infinitely many states. Then for every state L_x we have an accepting state L_{xx}, which we enter if the element x has been processed a second time. If we process elements that are not equal to x, we stay in the state L_x. From the accepting state L_{xx} it is fine we process more x's, but if we process something that is not x, then we go back to the previous state L_x. This notation of the state diagram is not very useful if we want to describe the whole automata.

A solution for this is to only give the states that represent the orbits of $\mathrm{Der}(L)$. In the above state diagram we can see that after one element has been processed, the rest of the diagram is basically the same. Since Q is a G-set, we can also draw the state diagram of M_L as follows:



This state diagram collapses all the states that are in the same orbit into single states, and by means of group actions, all of the states in the original automata can be obtained. This state diagram also gives a visualisation of the orbits of $\text{Der}(L)$. Each state in the collapsed state diagram represents an orbit. Since we are interested in drawing automata for nominal languages, for which it holds that $\text{Der}(L)$ is orbit finite, this is a nice way of depicting the state diagram.

Lemma 4.2.5. *The syntactical automata M_L recognizes the language L.*

Proof. For a word $w \in L$ we have that $\delta^*(q_0, w) = \delta^*(L, w) = L_w \in F$ by definition of the automata. This means for exactly the words w that are in L, $\delta^*(q_0, w)$ ends up in a final state, hence every exactly L recognized by M_L. □

Lemma 4.2.6. *Let X be an orbit finite nominal G-set. Let $L \subseteq X^*$ be an nominal language such that Der(L) is orbit finite. Then the syntactical automaton M_L is a nominal automaton.*

Proof. Assume that X is an orbit finite nominal G-set. Assume $L \subseteq X^*$ is a nominal language such that $\text{Der}(L)$ is orbit finite. We will now check that all the components of M_L are in line with definition 4.2.1.

- $Q = \text{Der}(L)$ should be an orbit finite nominal set. By lemma 4.1.9 we know that it is nominal, and by assumption it is orbit finite.
- $\Sigma = X$ should be an orbit finite nominal set. This holds, by assumption.
- $\delta(L_w, x) = L_{wx}$ should be an equivariant function. This means that $\delta(L_w, x) \cdot \pi = \delta(L_w \cdot \pi, x \cdot \pi)$ should hold. We see that this is indeed the case, by writing out $\delta(L_w, x) \cdot \pi$, and by using that $L_{w \cdot \pi} = L_w \cdot \pi$ (lemma 4.1.6):

  $$\delta(L_w, a)\cdot\pi = L_{wa}\cdot\pi = L_{(wa)\cdot\pi} = L_{(w\cdot\pi)(a\cdot\pi)} = \delta(L_{w\cdot\pi}, a\cdot\pi) = \delta(L_w\cdot\pi, a\cdot\pi).$$

- $I = L$ should be should be an equivariant subset of $Q = \text{Der}(L)$. Since L is a nominal language is is defined as an equivariant subset of X^*, hence L is equivariant. It is also a subset of $\text{Der}(L)$, since we can write $L = L_\varepsilon$, the derivative of the empty word ε.
- $F = \{L_w \mid w \in L\}$ should be an equivariant subset of $Q = \text{Der}(L)$. We know that L is equivariant. This means that for every $\pi \in G$ and for every $w \in L$ we have that $w \cdot \pi \in L$. From this we can deduce that the set $\{L_w \mid w \in L\}$ is equivariant by writing out $\{L_w \mid w \in L\} \cdot \pi$ and again using that $L_{w\cdot\pi} = L_w \cdot \pi$ (lemma 4.1.6):

  $$\{L_w \mid w \in L\}\cdot\pi = \{L_w\cdot\pi \mid w\cdot\pi \in L\} = \{L_{w\cdot\pi} \mid w\cdot\pi \in L\} = \{L_v \mid v \in L\}.$$

□

We recall lemma 2.2.3, which states that $L_w = \delta^*(q_0, w)$, for a regular language L, and machine M that recognizes L. This lemma also holds in the nominal case, since that proof only makes use of the definition of a derivative language, which is basically unchanged in the nominal setting.

We now have all the pieces to put together the theorem we mentioned at the beginning of this section. We will prove that for a nominal language L there is an equivalence between $\mathrm{Der}(L)$ being orbit finite, and L being recognized by a nominal DFA.

Theorem 4.2.7. *Let L be a nominal language. Then L is recognized by a nominal deterministic finite automaton if and only if Der(L) is orbit finite.*

Proof. First, assume that L is recognized by a nominal deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$. Then we have that

$$\begin{array}{rcl}
\mathrm{Der}(L) & = & \{L_w \mid w \in \Sigma^*\} \\
& = & \{\{w \mid vw \in L\} \mid w \in \Sigma^*\} \\
& = & \{\{w \mid \delta^*(q_0, vw) \in F\} \mid w \in \Sigma^*\} \\
& = & \{\{w \mid \delta^*(\delta^*(q_0, v), w) \in F\} \mid w \in \Sigma^*\} \\
& = & \{\mathcal{L}(\delta^*(q_0, v) \mid w \in \Sigma^*\} \\
& \subseteq & \{\mathcal{L}(q) \mid q \in Q\}
\end{array}$$

By definition of a nominal DFA we know the the set of stated Q is orbit finite. From this we conclude that there are orbit finitely many state languages $\mathcal{L}(q)$, and thus the set $\{\mathcal{L}(q) \mid q \in Q\}$ is orbit finite. Then we have seen in lemma 4.1.6 that $\mathrm{Der}(L)$ is an equivariat set. Now we can conclude from proposition 3.2.5 that $\mathrm{Der}(L)$ is a union of orbits, which come from the set $\{\mathcal{L}(q) \mid q \in Q\}$, hence $\mathrm{Der}(L)$ is orbit finite.

Conversely, assume that L is a nominal language such that $\mathrm{Der}(L)$ is orbit finite. The from lemma 4.2.6 we can immediately conclude that there is a nominal automaton that recognizes L, namely, the syntactical automata M_L. □

Chapter 5

Closure properties of nominal languages

Now that we have introduced nominal languages and nominal DFA's, we are going to look into their closure properties under substitutions. If we take a nominal language, and apply a substitution, will the result still be a nominal language? And if we assume that a nominal language is recognized by a nominal DFA, will the image of a substitution then still be recognized by a nominal DFA? To be able to answer such questions we first have to define what homomorphisms and substitutions are, in the nominal setting.

5.1 Nominal substitutions

Definition 5.1.1. Let A and B be orbit finite nominal G-sets. Take an equivariant function $h : A \to B^*$. We extend this function to $h^\sharp : A^* \to B^*$ such that the following property holds:

$$\forall a, b \in A^* \; [h^\sharp(ab) = h^\sharp(a)h^\sharp(b)]$$

where $h^\sharp(x) = h(x)$ for $x \in A$. A function $h^\sharp$ with this property is called a *nominal homomorphism.* Since $h^\sharp$ is derived from h, we also say that h is a nominal homomorphism. We see that $h^\sharp$ is now defined for words in A^*. Using the property, we see that following holds for $w \in A^*$:

$$h^\sharp(w) = h^\sharp(a_1a_2\ldots a_n) = h^\sharp(a_1)h^\sharp(a_2)\ldots h^\sharp(a_n) = h(a_1)h(a_2)\ldots h(a_n).$$

We remind ourselves what it means for a function f to be equivariant. This means that $f(x \cdot \pi) = f(x) = f(x) \cdot \pi$, for every $\pi \in G$. This would mean that for a function to be equivariant, we need that the image $f(x)$ of an element x, and x itself have to be supported by the same support C. In other words, the function needs to preserve support.

Example 5.1.2. Take the sets $A = B = \mathbb{A}$, the atoms. We define a homomorfism $h : \mathbb{A} \to \mathbb{A}^*$ by $h : a \mapsto a^n$ for $a \in \mathbb{A}$ and $n \in \mathbb{N}$. This is an equivariant function, since the support $C = a$ is a support for both a and $h(a)$. From this definition of h we get $h^\sharp : \mathbb{A}^* \to \mathbb{A}^*$, where if we write for $w \in \mathbb{A}^*$ that $w = a_1 a_2 \ldots a_k$ then $h^\sharp : w \mapsto a_1^{n_1} a_2^{n_2} \ldots a_k^{n_k}$ for $n_i \in \mathbb{N}$. Here again, if C supports w, it also supports $h^\sharp(w)$, since the same atoms occur in both.

Definition 5.1.3. Let A and B be orbit finite nominal G-sets. Take an equivariant function $s : A \to \mathcal{P}_{fs}(B^*)$. We extend this function to $s^\sharp : A^* \to \mathcal{P}_{fs}(B^*)$ such that the following property holds:

$$\forall a, b \in A^* \; [s^\sharp(ab) = s^\sharp(a) s^\sharp(b)]$$

where $s^\sharp(x) = s(x)$ for $x \in A$. If $s^\sharp(a)$ and $s^\sharp(b)$ are both sets, $s^\sharp(a)s^\sharp(b)$ contains all words vw such that $v \in s^\sharp(a)$ and $w \in s^\sharp(b)$. A function $s^\sharp$ with the above property is called a *nominal homomorphism.* Since $s^\sharp$ is derived from s, we also say that s is a nominal substitution. We see that $s^\sharp$ is now defined for words in A^*. Using the property, we see that following holds for $w \in A^*$:

$$s^\sharp(w) = s^\sharp(a_1 a_2 \ldots a_n) = s^\sharp(a_1) s^\sharp(a_2) \ldots s^\sharp(a_n) = s(a_1) s(a_2) \ldots s(a_n).$$

Example 5.1.4. Take the sets $A = B = \mathbb{A}$, the atoms. We then define the substitution $s : \mathbb{A} \to \mathcal{P}(\mathbb{A}^*)$ as $s : a \mapsto \mathbb{A}^* \backslash \{a\}$. We note that a and $s(a)$ have the same support $C = \{a\}$. With the extension, we get $s^\sharp : \mathbb{A}^* \to \mathcal{P}_{fs}(\mathbb{A}^*)$, such that for a word $w = a_1 a_2 \ldots a_k \in \mathbb{A}^*$, we get $s^\sharp : w \mapsto \mathbb{A}^* \setminus \{a_1 a_2 \ldots a_k\}$.

A nominal substitution s is an equivariant function by definition, but for an extension, this is not immediately clear. The following lemma shows us that an extension of a nominal substitution is also equivariant.

Lemma 5.1.5. *Let $s^\sharp : A^* \to \mathcal{P}_{fs}(B^*)$ be a nominal substitution derived from s. Then $s^\sharp$ is equivariant, i.e. $s^\sharp(w) \cdot \pi = s^\sharp(w \cdot \pi)$.*

Proof. Let $s^\sharp : A^* \to \mathcal{P}_{fs}(B^*)$ be a nominal substitution derived from s. By writing out $s^\sharp(w) \cdot \pi$ we get:

$$\begin{aligned}
s^\sharp(w) \cdot \pi &= s^\sharp(a_1 a_2 \ldots a_n) \cdot \pi && \text{(Write out } w\text{)} \\
&= (s(a_1) s(a_2) \ldots s(a_n)) \cdot \pi && \text{(Definition of } s^\sharp\text{)} \\
&= (s(a_1) \cdot \pi)(s(a_2) \cdot \pi) \ldots (s(a_n) \cdot \pi) && \text{(Application of } \pi\text{)} \\
&= s(a_1 \cdot \pi) s(a_2 \cdot \pi) \ldots s(a_n \cdot \pi) && \text{(Equivariance of } s\text{)} \\
&= s^\sharp((a_1 \cdot \pi)(a_2 \cdot \pi) \ldots (a_n \cdot \pi)) && \text{(Definition of } s^\sharp\text{)} \\
&= s^\sharp((a_1 a_2 \ldots a_n) \cdot \pi) && \text{(Application of } \pi\text{)} \\
&= s^\sharp(w \cdot \pi) && \text{(Write back } w\text{)}
\end{aligned}$$

□

Corollary 5.1.6. *Let $s^\sharp : A^* \to \mathcal{P}_{fs}(B^*)$ be the extension of a substitution s. Then $s^\sharp$ preserves support.*

Proof. From lemma 3.3.8 we know that equivariant functions preserve support. In lemma 5.1.5 we have seen that $s^\sharp$ is an equivariant function, hence it preserves support. □

5.2 Closure properties

We have introduced nominal languages. We have introduced nominal substitutions. This means that we can now finally investigate the question: "Are nominal languages closed under nominal subtitutions?". We will see in the following theorem that this is the case!

Theorem 5.2.1. *Nominal sets are closed under substitution.*

Proof. Let A and B be orbit finite nominal G-sets. Let L be a nominal language, such that $L \subseteq A^*$. Let $s^\sharp : A^* \to \mathcal{P}_{fs}(B^*)$ be a substitution. Then we want to prove that $s^\sharp(L)$ is a nominal language, which means that $s^\sharp(L)$ is an equivariant subset of B^*. This means that

$$\forall \pi \in G[s^\sharp(L) \cdot \pi = s^\sharp(L)].$$

We write out:

$$\begin{aligned}
s^\sharp(L) \cdot \pi &= (\textstyle\bigcup_{w \in L} s^\sharp(w)) \cdot \pi \\
&= (\textstyle\bigcup_{w \in L} s^\sharp(w) \cdot \pi) \\
&= (\textstyle\bigcup_{w \in L} s^\sharp(w \cdot \pi) && \text{(lemma 5.1.5 : equivariance of } s^\sharp) \\
&= (\textstyle\bigcup_{v \in L} s^\sharp(v) && (L \text{ is equivariant, so } w \cdot \pi \in L) \\
&= s^\sharp(L) && \text{(Indexing over } v \text{ also gives } L)
\end{aligned}$$

We conclude that $s^\sharp(L)$ is an equivariant subset of B^*, and thus it is a nominal language. □

The above theorem yields a nice result. With theorem 4.2.7 (L recognized by a nominal DFA iff $\mathrm{Der}(L)$ is orbit finite), we would like to prove that if L is recognized by a nominal DFA, that the image $s^\sharp(L)$ is also recognized by a nominal DFA. It turns out that this is not the case, as we can give a counterexample which shows the opposite.

Example 5.2.2. Take the language $L = \{aa \mid a \in \mathbb{A}\}$, This is a nominal language, since every element $aa \in L$ is supported by $C = \{a\}$. Also $\mathrm{Der}(L)$ is orbit finite, it has the orbits represented by L, L_a, and L_{aa}. From theorem 4.2.7 we can conclude that L is recognized by a nominal DFA. Then take the substitution $s(a) = \{wav \mid w, v \in \mathbb{A}^*, a \text{ does not occur in } v, w\}$. We then look at the image of L under the extension of this substitution $s^\sharp(L) = \{wavau \mid w, v, u \in \mathbb{A}^*, a \in \mathbb{A}\} = \{w \mid \text{ some atom exactly occurs twice}\}$. This language can not be recognized by a nominal DFA, since it has orbit infinitely many derivatives. This is shown by the following derivatives which all have a separate orbit:

$$s^\sharp(L)_a = \{vau \mid w, v \in \mathbb{A}^*, a \text{ does not occur in } v, u\}$$

$$s^\sharp(L)_{ab} = \{vau \mid w, v \in \mathbb{A}^*, a \text{ does not occur in } v, u\} \cup$$
$$\{vbu \mid w, v \in \mathbb{A}^*, b \text{ does not occur in } v, u\} = s^\sharp(L)_{ba}$$
$$s^\sharp(L)_{abc} = \{vau \mid w, v \in \mathbb{A}^*, a \text{ does not occur in } v, u\} \cup$$
$$\{vbu \mid w, v \in \mathbb{A}^*, b \text{ does not occur in } v, u\} \cup$$
$$\{vcu \mid w, v \in \mathbb{A}^*, c \text{ does not occur in } v, u\} = s^\sharp(L)_{cba} = s^\sharp(L)_{bac} = ...$$
$$\vdots$$

We see that for instance, there are no words of the from the set $\{vbu \mid w, v \in \mathbb{A}^*, b \text{ does not occur in } v, u\}$ in $s^\sharp(L)_a$. This means that $s^\sharp(L)_{ab} \neq s^\sharp(L)_a$. In a similar way, all of these derivatives can be proven in-equal. Since a group action does not change the amount of atoms in a word, all these derivatives are in separate orbits. Now, since $\mathbb{A}$ is infinite, the set $\mathrm{Der}(s^\sharp(L))$ is orbit infinite. By theorem 4.2.7 we then get that $s^\sharp(L)$ is not recognized by a nominal DFA.

Chapter 6

Related Work

A big inspiration for the theory in this thesis was that of Bojańczyk et al [2]. In Bojańczyk's paper, the order in which lemma's are proven differs from our work, and some definition have different notation and names. This shows that the work we have given has been worked out without explicitly copying Bojańczyk's work. We also work out some theorems in more detail, and omit less steps in proofs, which makes this thesis a nice contribution.

Then there is work of Moerman et al [5]. In this paper Moerman looks into different classes of languages, which are distinguished by properties of their derivatives. For one of these classes, the residual languages, a learning algorithm is described. Moerman's paper was the main inspiration for this thesis, since it discussed the nominal languages, and made us wonder about their closure properties. The main point of Moerman's paper is not to investigate nominal languages, and does not explicitly discuss closure properties of nominal languages. To that point, this thesis gives a nice piece of background information for such work.

Chapter 7

Conclusions

In this thesis, the closure properties of nominal languages under substitutions have been explored. A big part of this was defining the proper theory to be able to properly discuss nominal languages. First we defined some basic notions in automata theory. We then gave a proof for a Myhill-Nerode style theorem, regarding the construction of an automaton with derivative languages. We took Shallit's [7] proof, altered it into a separate lemma and theorem, and added some details about induction that were skipped in the original proof. Then we defined some group theory, involving orbits and equivariance[4]. From these notions we defined nominal sets[2].

For nominal sets we have shown that the Myhill-Nerode style theorem also holds. We studied the work of Bojańczyk et al. [2], from this work we took some theorems which we proved in our own framework. We proved that nominal language are recognized by a nominal deterministic finite automata if and only if the set of all of its derivatives is orbit finite. We also proved that nominal languages are indeed closed under substitutions. When trying to combine the Myhill-Nerode theorem, and the closure property we found a counter example, which showed that being recognized by a nominal DFA, is not closed under nominal substitutions. This final counter-example is a negative result, but that does not matter for the over-all result of this thesis. Along the way a lot of nice theory has been discussed, and nice theorems have been proven. Next to that, the result that being recognized by a nominal DFA is not preserved by nominal substitutions, is a nice result regardless.

For future work we could look into closure properties under substitutions for other classes of languages, for example the so-called residual languages [5]. To be able to define such languages, the field of lattice theory has to be used. This is another field in mathematics which studies certain types of orderings, called lattices. This is a very interesting direction to continue this research, but did not fit the time constraints of this thesis.

Bibliography

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106, 1987.

[2] Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014.

[3] Janet Chen. *Group theory and the Rubik's cube.* 2004.

[4] Ben Moonen Hendrik Lenstra Jr., Frans Oort. *Groepentheorie.* lecture notes. Radboud Universiteit, NL, 2014.

[5] Joshua Moerman and Matteo Sammartino. Residual nominal automata. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 44:1–44:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[6] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 613–625, New York, NY, USA, 2017. Association for Computing Machinery.

[7] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press, USA, 1 edition, 2008.

[8] Thomas A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science.* Addison-Wesley Longman Publishing Co., Inc., USA, 1997.