Radboud University Nijmegen

Faculty of Science

On iterated transducers and closure under hypotheses

Discussing the hidden relationship

Thesis BSc Computing Science

Author:
Robin Holen

Supervisor:
dr. Jurriaan Rot

Second reader:
prof. dr. Herman Geuvers

April 2022

Abstract

Hypotheses are inequations of the form $e \leq f$ where e and f are regular expressions. Transducers and languages closed under hypotheses (H-closure) define functions from language to language. We can iterate such transducers to create new transducers. Taking the union of all finite iterations yields the ω-iteration. The H-closure is by definition an ω-iterated function. Comparing the ω-iterted transducer and the H-closure reveals that these two notions are the same under specific circumstances. Given a transducer T, we change the transducer to T_c, and then we can make a set of hypotheses H such that the ω-iteration of T_c coincides with the H-closure. Also, given a set of hypothesis H with inequations of the form $e \leq w$ where e is a regular expression and w a word, we can construct a transducer such that its ω-iteration coincides with the closure under H.

Contents

Chapter 1

Introduction

Typically, computing science undergraduate students encounter languages and automata early in their education. The world of formal language and automata theory is vast and profound and cannot be all taught within one course. Automata are abstract machines. Machines that can be used to solve computational problems. Aside from the computational aspect of automata, these abstract machines are also important in areas such as compiler construction or text processing [7]. Also, there have been uses outside of computing science, such as in bioscience [2]. The many fields where automata theory is applicable are vast and cannot be contained in only one course. We focus on one kind of automata within the automata theory called a transducer.

Transducers are finite automata that map formal languages to other formal languages. We can see a transducer as a finite state automaton with output. The research behind the transducers traces back to at least 1965 [6]. Since then, transducers have become part of several "advanced" courses on automata theory such as "*A second course in formal languages and automata theory*" from J. Shallit [10].

We also have a different notion called H-closures, closures for short. *Closures* are defined by applying hypotheses to a language. For instance, apply $ab \leq ba$ to the language of the regular expression b^*a^* (denoted as $[\![b^*a^*]\!]$). We apply $ab \leq ba$ to $[\![b^*a^*]\!]$, meaning we can rewrite every instance of ba to ab as many times as we can, and we get a new language $[\![(a+b)^*]\!]$. In general, hypotheses take the form of an inequation, "$e \leq f$", where e, f are both regular expressions [5].

One may think there are some similarities between these two notions. They both, in some sense, have a language, to begin with, and after using the transduction/closure, there is a (different) language. Transducers, the machines that perform the transductions, follow a transition relation to map languages to languages. H-closures use the given hypotheses to determine the language closed under these hypotheses.

The method these notions use to obtain languages is somewhat different, and there is no trivial way to see if transducers and closures are even related. Nevertheless, the similarities they do have, suggest that there is a relation between the two.

In this thesis, we uncover that relation. The contributions made in this thesis rely on ω-iteration. It is possible to iterate transducers. Finite iterations of transducers preserve regularity. Iteration extends to iterating a transducer countably many times. Such iteration is called the ω-iteration of said transducer. We know that the ω-iteration does not preserve regularity in general [8]. Nonetheless, ω-iteration proved to be helpful.

This thesis covers two theorems about this relation. First, given a transducer T, we can create a slightly different transducer T_c. From this new transducer, we make a set of hypotheses H so that the ω-iteration of T_c coincides with the H-closure. This T_c is almost the same as T; the difference lies in that T_c can return the input word as output while T does not necessarily have such a possibility. Second, given a set of hypotheses H where each hypothesis is of the form $e \leq w$ with e a regular expression and w a word, we make a transducer T such that the ω-iteration of T coincides with the H-closure. We cannot miss the requirement that each hypothesis is in the form $e \leq w$. Transducers can only read words, not entire languages. Hence it does not make sense to allow hypotheses of the form $e \leq f$ with f a regular expression.

We will cover details of the results after we have introduced both notions properly. First, we need to cover some preliminaries. Chapter 2 contains general information about finite automata and regular expressions that many undergraduate students may see in their first year. In chapter 3, we take a look at transducers themselves. Here we precisely define transducers and give basic properties such as regularity preservation and composition. All these properties build to the definition of iteration, particularly ω-iteration. Here we will show that ω-iteration does not preserve regularity by giving a concrete counterexample.

In chapter 4, we give a formal definition of what H-closures are and how we can use them to determine what the closure will be. For this, we use a proof system defined in [5] which will help prove what the closures are. Another tool will be transfinite induction. Chapter 5 is the chapter where we prove the relation between closures and transducers. We show two equalities, one assuming we have a set of assumptions and another assuming we have a transducer. The chapter concludes with a discussion on the use of closures and transducers. The discussion will contain the pros and cons of each method to convey in what situations which method is best to use.

Chapter 2

Preliminaries

This section will describe the notions of regular languages, regular expressions, and finite automata. These are standard notions. See, for instance, [10].

2.1 Regular languages and regular expressions

Definition 2.1.1. An alphabet Σ is a finite set of symbols.

Definition 2.1.2. A word over an alphabet Σ is a finite sequence $a_1 a_2 \dots a_n$ where for each $i \leq n$ we have $a_i \in \Sigma$. There also exists the empty word denoted as ε. If w is a word then $\varepsilon w = w\varepsilon = w$. Furthermore, Σ^* is the set of all words and is inductively defined as:

1. $\varepsilon \in \Sigma^*$
2. if $x \in \Sigma$ and $v \in \Sigma^*$ then $xv \in \Sigma^*$

We denote the length of a word w by $|w|$ and the number of occurences of the letter $x \in \Sigma$ by $|w|_x$.

Definition 2.1.3. A formal language L over an alphabet Σ is a subset of Σ^*

Examples of languages are the Dutch language (which has the widely known usual alphabet), the Japanese language, the programming language C++, the Klingon language, and the Python programming language. These languages are vibrant and extensive, which goes far beyond the purpose of this thesis. We will focus more on formal languages. Examples of such languages are the language with an arbitrary number of repetitions of ab, the language of all binary numbers, or even the language of all words with a specific subword. These languages are written as $\{(ab)^n \mid n \in \mathbb{N}\}, \{0,1\}^*$ and $\{w \in \Sigma^* \mid v \in \Sigma^* \text{ is a subword of } w\}$. One should note what the meaning is of $\{(ab)^n \mid n \in \mathbb{N}\}$. The letters a and b are not numbers, so "exponentiation" is not defined.

We, therefore, need to define what "exponentiation" and other operations on languages are. Suppose $L_1, L_2, L \subseteq \Sigma^*$ are languages, then the following are also languages:

$$\begin{aligned} L_1 \cup L_2 &= \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\} \\ L_1 \cap L_2 &= \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\} \\ \overline{L} &= \{w \in \Sigma^* \mid w \notin L\} \\ L_1 L_2 &= \{w_1 w_2 \in \Sigma^* \mid w_1 \in L_1 \wedge w_2 \in L_2\} \\ L^0 &= \{\varepsilon\} \\ L^{n+1} &= LL^n \\ L^* &= \bigcup_{n \in \mathbb{N}} L^n \end{aligned}$$

This shows that $(ab)^n \neq a^n b^n$ since

$$\begin{aligned} \{(ab)^n \mid n \in \mathbb{N}\} &= \{\varepsilon, ab, abab, ababab, \dots\} \\ &\neq \{\varepsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\} \end{aligned}$$

Formal languages are classified into different classes of languages. One such class is the class of regular languages. Regular languages are an essential class of languages. They are used in string processing and lexical analysis in programming language compilation [1]. Besides regular languages, there are also related regular expressions.

Definition 2.1.4. Regular expressions are inductively defined over Σ as follows:

- 0, 1 and $a \in \Sigma$ are regular expressions.
- If r, s are regular expressions, then so are $(r + s), rs, r^*$

With $[\![r]\!]$ we denote the language of r which is defined as:

- $[\![0]\!] = \varnothing$
- $[\![1]\!] = \{\varepsilon\}$
- $[\![a]\!] = \{a\}$
- $[\![rs]\!] = [\![r]\!][\![s]\!]$
- $[\![r + s]\!] = [\![r]\!] \cup [\![s]\!]$
- $[\![r^*]\!] = ([\![r]\!])^*$

We say a language L is regular if there is an regular expression e such that $L = [\![e]\!]$. For example, consider $\{(ab)^n \mid n \in \mathbb{N}\}$. This language has the expression $(ab)^*$ because

$$[\![(ab)^*]\!] = [\![ab]\!]^* = \{ab\}^* = \{\varepsilon, ab, abab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$$

Another example is $b^*(abba + baab)b^*$. The corresponding language is:

$$\begin{aligned}[\![b^*(abba + baab)b^*]\!] &= [\![b^*]\!]([\![abba]\!] \cup [\![baab]\!])[\![b^*]\!] \\ &= \{\varepsilon, b, bb, bbb, \dots\}\{abba, baab\}\{\varepsilon, b, bb, bbb, \dots\}\end{aligned}$$

2.2 Automata

We turn to the notion of automata, abstract machines related to languages and regular languages. The focus will primarily be on finite state automata. There are many different kinds of finite-state automata, and we will introduce three kinds.

Definition 2.2.1. A deterministic finite automaton (DFA) is a tuple $M = \langle Q, \Sigma, q_0, \delta, F\rangle$ where:

- Q is a finite set of states
- Σ is an alphabet
- q_0 is the initial state
- $\delta : Q \times \Sigma \to Q$ is a transition function
- $F \subseteq Q$ is the set of accepting states

It is important to note that to accept words (i.e., we are in an accepting state), we need a transition function that can read words and then output a state. Here the multi-step transition $\delta^* : Q \times \Sigma^* \to Q$ is inductively defined as:

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

With this function, we can input whole words. The language accepted by the DFA M is the set $\mathcal{L}(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$.

Figure 2.1 is a DFA that accepts any word with the substring aba in it. Let us demonstrate how this works. Suppose we have the word $aabab$.

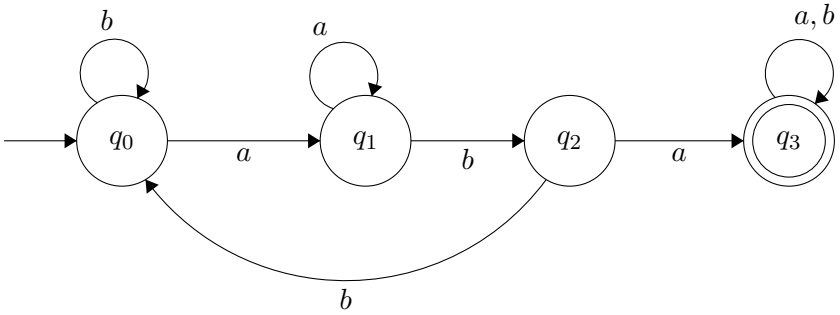

Figure 2.1: A DFA that accepts any word with the substring *aba* in it.

The initial state of our DFA is q_0. We start with computing $\delta^*(q_0, aabab)$.

$$\begin{aligned}
\delta^*(q_0, aabab) &= \delta^*(\delta(q_0, a), abab) &&= \delta^*(q_1, abab) \\
&= \delta^*(\delta(q_1, a), bab) &&= \delta^*(q_1, bab) \\
&= \delta^*(\delta(q_1, b), ab) &&= \delta^*(q_2, ab) \\
&= \delta^*(\delta(q_2, a), b) &&= \delta^*(q_3, b) \\
&= \delta^*(\delta(q_3, b), \varepsilon) &&= \delta^*(q_3, \varepsilon) \\
&= q_3
\end{aligned}$$

Thus we see that $\delta^*(q_0, aabab) = q_3$. Since q_3 is an accepting state, the word *aabab* is accepted by the DFA. In the same fashion we can work out if *abbba* is accepted by this DFA. We start with computing $\delta^*(q_0, abbba)$.

$$\begin{aligned}
\delta^*(q_0, abbba) &= \delta^*(\delta(q_0, a), bbba) &&= \delta^*(q_1, bbba) \\
&= \delta^*(\delta(q_1, b), bba) &&= \delta^*(q_2, bba) \\
&= \delta^*(\delta(q_2, b), ba) &&= \delta^*(q_0, ba) \\
&= \delta^*(\delta(q_0, b), a) &&= \delta^*(q_0, a) \\
&= \delta^*(\delta(q_0, a), \varepsilon) &&= \delta^*(q_1, \varepsilon) \\
&= q_1
\end{aligned}$$

The state q_1 is not an accepting state; hence the machine does not accept the word *abbba*.

DFAs have the lovely property that they are deterministic (as the name says), which means that if we run a DFA twice by inputting the same word, the results will be equal. There are also non-deterministic finite-state automata. That is, there are multiple transitions possible from one state for a given letter.

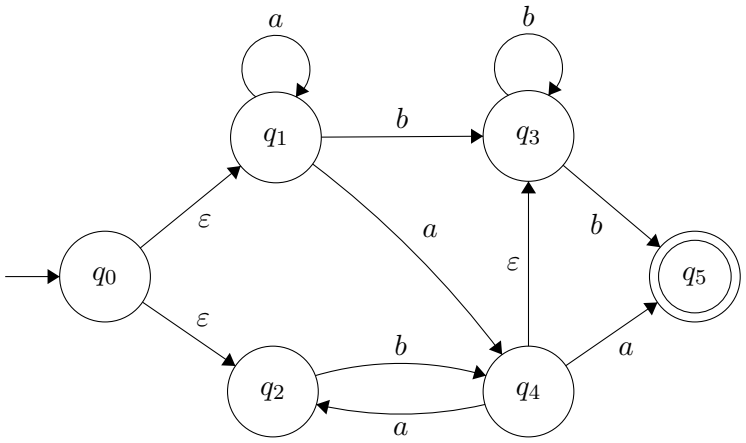

Figure 2.2: An NFA-ε. Here we can see that ε-closure$(q_0) = \{q_0, q_1, q_2\}$. Note here that every state has an implicit ε transition to itself.

Definition 2.2.2. A non-deterministic finite automaton with ε transitions (NFA-ε) is a tuple defined as a DFA except for the transition function. The transition function is a function $\delta : Q \times (\Sigma \cup \varepsilon) \to \wp(Q)$ which takes in a state and a symbol and returns a set of states. Here $\wp(Q)$ denotes the power set of Q. The ε-closure(q) is the set of states reachable from q with only ε-steps. The multi step transition function is defined as:

- $\delta^*(q, \varepsilon) = \varepsilon$-closure$(q)$
- $\delta^*(q, aw) = \bigcup_{q' \in \varepsilon\text{-closure}(q)} \bigcup_{p \in \delta(q', a)} \delta^*(p, w)$

An NFA is an NFA-ε but without the ε transitions. The language accepted by the NFA(-ε) is the set $\{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \varnothing\}$.

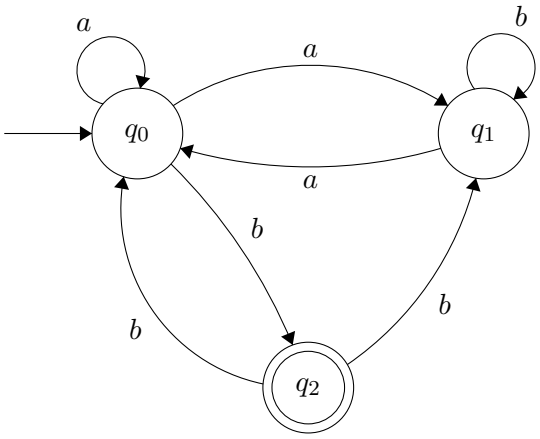

Figure 2.3: An NFA. If we read an a while in state q_0 we can go to either q_1 or to q_0. Thus $\delta(q_0, a) = \{q_0, q_1\}$.

Figure 2.2 depicts an NFA-ε. Let us demonstrate the workings of this NFA-ε. We want to check if the NFA-ε accepts $aabb$.

$$
\begin{aligned}
\delta^*(q_0, aabb) &= \bigcup_{q' \in \varepsilon\text{-closure}(q_0)} \bigcup_{p \in \delta(q', a)} \delta^*(p, abb) \\
&= \bigcup_{p \in \delta(q_0, a)} \delta^*(p, abb) \cup \bigcup_{p \in \delta(q_1, a)} \delta^*(p, abb) \cup \bigcup_{p \in \delta(q_2, a)} \delta^*(p, abb) \\
&= \delta^*(q_1, abb) \cup \delta^*(q_4, abb) \\
&= \bigcup_{q' \in \varepsilon\text{-closure}(q_1)} \bigcup_{p \in \delta(q', a)} \delta^*(p, bb) \cup \bigcup_{q' \in \varepsilon\text{-closure}(q_4)} \bigcup_{p \in \delta(q', a)} \delta^*(p, bb) \\
&= \bigcup_{p \in \delta(q_1, a)} \delta^*(p, bb) \cup \bigcup_{p \in \delta(q_3, a)} \delta^*(p, bb) \cup \bigcup_{p \in \delta(q_4, a)} \delta^*(p, bb) \\
&= \delta^*(q_1, bb) \cup \delta^*(q_2, bb) \cup \delta^*(q_4, bb) \cup \delta^*(q_5, bb) \\
&= \dots \\
&= \delta^*(q_3, \varepsilon) \cup \delta^*(q_5, \varepsilon) \\
&= \{q_3, q_5\}
\end{aligned}
$$

For clarity, we leave out a part of the computation since it mostly the same kind of computation in the first few steps but bigger and less readable.

Earlier, we stated that finite-state automata are related to regular languages. Finite-state automata accept languages. A famous result by Stephen Cole Kleene is the following:

Theorem 2.2.3 (**Kleene's theorem**). *The class of languages accepted by DFAs is the class of regular languages.*

The theorem tells us that a DFA accepts that language for every regular language. Proving the theorem can be done with the notions of NFA and NFA-ε. Given a regular language, we have a regular expression from which we can create an NFA-ε. An NFA-ε can then be transformed into an NFA, allowing us to construct a DFA. Lastly, with a DFA, we can construct the regular expression accepted by the DFA.

Chapter 3

Transducers

Transducers are a form of generalized finite-state automata with output. Conceptually, we can see them as formal language translators. The definitions are based on the theory as described in [10].

3.1 Definition and properties

Definition 3.1.1. A finite state transducer T is a 6-tuple $\langle Q, q_0, \Sigma, \Gamma, R, F \rangle$ where:

- Q is a finite set of states.
- q_0 is the initial state.
- Σ is the input alphabet.
- Γ is the output alphabet.
- $R \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ is a transition relation.
- $F \subseteq Q$ is the set of final states.

An element $\langle q_i, x, y, q_j \rangle \in R$ is called a rule. The multistep relation $R^\#$ is the smallest relation such that:

- $R \subseteq R^\#$
- if $\langle q_i, w_1, v_1, q_j \rangle$ and $\langle q_j, w_2, v_2, q_k \rangle \in R^\#$ then $\langle q_i, w_1 w_2, v_1 v_2, q_k \rangle \in R^\#$

Furthermore, we write a rule $\langle q_i, x, y, q_j \rangle \in R$ as $q_i x \rightarrow_T y q_j$. We write $q_i x \rightarrow y q_j$ if it is clear from the context that the transition rule belongs to T. Similarly, we write $\langle q_n, w, v, q_m \rangle \in R^\#$ as $q_n w \rightarrow^\# v q_m$. The transition $\rightarrow^\#$ allows for stitching rules together and is called the reflexive transitive closure.

Suppose we have $T = \langle\{\{q_0\}, q_0, \{a\}, \{b\}, \{q_0a \to q_0b\}, \{q_0\}\}$.
A transduction of aaa uses the rule three times, for which we get the sequence $q_1aaa \to bq_1aa, bq_1aa \to bbq_1a$ and $bbq_1a \to bbbq_1$. Yet with the transitive closure, we have that

$$q_1aaa \to bq_1aa \to bbq_1a \to bbbq_1$$

is just the same transduction as

$$q_1aaa \to^{\#} bbbq_1$$

Finite-state transducers (transducers for short) are functions from words to the set of languages. If T is a transducer with input alphabet Σ and output Γ, then we can define $f_T : \Sigma^* \to \wp(\Gamma^*)$, the function that maps words to a language. Transducers can only read one word at a time; therefore, the language accepted by a transducer is the union of all the translated words. To put it more formally:

Definition 3.1.2. Let L be a language over Σ and $T = \langle Q, q_0, \Sigma, \Gamma, R_T, F\rangle$ a transducer. Let $w \in L$. Define $f_T : \Sigma^* \to \wp(\Gamma^*)$ as

$$w \mapsto \{v \mid \text{There exists a rule in } R_T^{\#} \text{ such that } q_0w \to^{\#} vq_f\}$$

where q_f is a final state. Furthermore,

$$f_T(L) = \bigcup_{w \in L} f_T(w)$$

If $S = \langle P, q_0', \Sigma, \Gamma, R_S, G\rangle$ is a transducer, then we say that S is equivalent to T ($S \sim T$) if and only if for any language L over Σ, $f_S(L) = f_T(L)$.

How does such a transducer work? Take a look at figure 3.1. In this case, our input and output alphabet are both $\{a, b\}$. Suppose we want a transduction of the word $abbba$. Every arrow in the transducer is a rule in R. A transduction of the word $abbba$ is effectively the same as that there exists a rule $q_0abbba \to^{\#} vq_i$ for a $v \in \{a, b\}^*$ and $i \in \{1, 2\}$.

$$q_0abbba \to q_1bbba \to bq_2bba \to baq_2ba \to baaq_2a \to baabq_1$$

These rules can all be combined to one rule in $R^{\#}$. For instance combining the first two rules, $q_0a \to q_1, q_1b \to bq_2$, becomes $q_0ab \to^{\#} bq_2$. When we keep on going, $q_0ab \to^{\#} bq_2$ and $q_2b \to_T aq_2$ becomes $q_0abb \to_T^{\#} baq_2$. Repeating the process will result $q_0abbba \to^{\#} baabq_1$ and so $\langle q_0, abbba, baab, q_1\rangle \in R^{\#}$. As demonstrated, $R^{\#}$ allows us to combine rules such that we can actually use the transducer. With this, it would be nice if we can also break rules apart into "smaller" rules. For this property, we define the notion of transducer normal form.

3.1.1 Transducer normal form

Definition 3.1.3. A transducer is in Transducer Normal Form (TNF) if and only if for every rule $\langle q_i, x, y, q_j \rangle \in R$ we have that $|x| \leq 1$ and $|y| \leq 1$.

The transducer in figure 3.1 is in TNF. It is desirable if every transducer has a TNF or, formally, if there exists a transducer T' that is in TNF such that for any language L, $f_T(L) = f_{T'}(L)$. Fortunately, this is possible. We designed an algorithm to create a new transducer with the TNF property.

Algorithm 1 Rewrite a rule into several other rules

```
1:  function RewriteRule(⟨q_i, x, y, q_j⟩)
2:      R' ← {⟨q_i, x, y, q_j⟩}
3:      r ← ∅
4:      while R' ≠ ∅ do
5:          rule ← EXTRACT(R)  ▷ Gets a rule from R and removes this rule from R
6:          x ← rule[1]
7:          y ← rule[2]
8:          p ← max(|x|, |y|)
9:          if 1 < p then
10:             a ← x[0]                          ▷ The first letter of x
11:             b ← y[0]                          ▷ The first letter of y
12:             w ← x[1 →]                        ▷ x without the first letter
13:             v ← y[1 →]                        ▷ y without the first letter
14:             Create new state q'_i
15:             Create new rule ⟨q_i, a, b, q'_i⟩
16:             r ← r ∪ {⟨q_i, a, b, q'_i⟩}
17:             Create new rule ⟨q'_i, w, v, q_j⟩
18:             R' ← R' ∪ {⟨q'_i, w, v, q_j⟩}
19:         else
20:             R' ← ∅
21:             r ← r ∪ {⟨q_i, x, y, q_j⟩}
22:         end if
23:     end while
24:     return r
25: end function
```

Here the assumption is made that if one of the words is empty (ε), then $rule[m][n]$ will just return ε regardless of m and n. The algorithm takes the first letter of each word x and y, and then puts that into a new rule. This new rule must contain a new state. If we did not create a new state, we would have created a new transducer. After creating the first rule, we need to create another rule since after combining the rules in r, we must end in q_j.

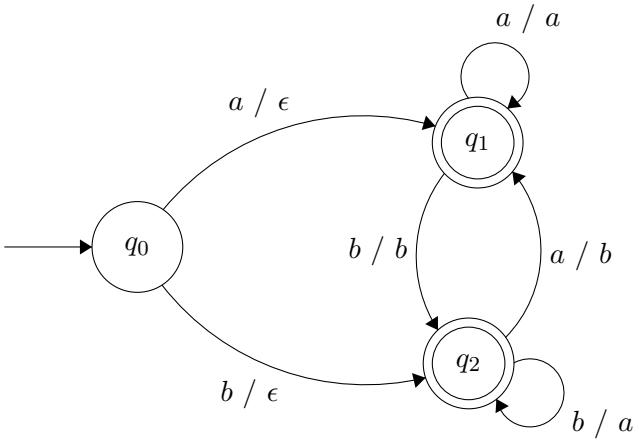

Figure 3.1: A finite-state transducer

We thus make the rule $q_i'w \to_T vq_j$. The set r now contains $q_ia \to bq_i'$ and R' contains $q_i'w \to_T vq_j$. Repeat creating new rules until both v and w are 1-, or 0-letter words. Since rules are transitive, all the rules in r used after each other will give the original rule. Every rule in r now only transduces letters to letters.

Proposition 3.1.4. *Let L be a language. Every finite-state transducer $T = \langle Q, q_0, \Sigma, \Gamma, R_T, F\rangle$ can be rewritten to transducer $S = \langle P, q_0, \Sigma, \Gamma, R_S, F\rangle$ that is in TNF such that $T \sim S$.*

Proof. Let T be a finite state transducer. Then for each element in R_T we use algorithm 1 to get the set R_S of rules. Now define $P = Q \cup \{q \mid q = \pi_4(r) \wedge r \in R_S\}$. In words, we take the states in Q and the states that we need extra to be able the create the rule r. Define the new transducer as $S = \langle P, q_0, \Sigma, \Gamma, R_S, F\rangle$. We are left to show that $f_T = f_S$. Important to note is that $R_T^{\#} = R_S^{\#}$ since R_S only contains rules from R_T that the algorithm made. It is therefore evident that $f_T = f_S$. □

The TNF property is rather nice in that it lets us assume that every rule in R only translates letters. If not, we can create an equivalent transducer that is in TNF.

3.1.2 Regularity preservation of transducers

Earlier, we discussed regular languages. Not all languages are regular such as $\{a^nb^n \mid n \in \mathbb{N}\}$. There does not exist a DFA that can accept that language. Regularity is a pleasant property to have for a language. Transductions preserve the regularity of a language.

Theorem 3.1.5. *Let L be a regular language over Σ and $T = \langle P, p_0, \Sigma, \Gamma, S, G\rangle$ a transducer. Then $f_T(L)$ is a regular language over Γ.*

Proof. L is regular thus there exists a DFA $M = \langle Q, q_0, \Sigma, \delta, F\rangle$ such that $\mathcal{L}(M) = L$ (where $\mathcal{L}(M)$ denotes the language accepted by M). To show that $f_T(L)$ is a regular language over Γ, we will construct an NFA-ϵ accepting $f_T(L)$. The NFA-ϵ M' will be constructed as follows:

- The set of states will be $Q \times P$.
- (q_0, p_0) will be the initial state.
- Γ is the alphabet.
- $F \times G$ will be the set of accepting states.
- $\delta' : (Q \times P) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times P)$ is the transition function that is defined as:

$$\delta'((q_i, p_j), y) = \{(\delta(q_i, x), p_k) \mid (p_j, x, y, p_k) \in S \wedge x \in \Sigma \cup \{\varepsilon\}\}$$

It is time to prove that $\mathcal{L}(M') = f_T(L)$.

- Suppose $v \in \mathcal{L}(M')$. This means that $\delta'^*((q_0, p_0), v) \cap F \neq \varnothing$. Assume $v = y_0 y_1 y_2 \dots y_n$ with $\forall_i[y_i \in \Gamma \cup \{\epsilon\}]$ (it can be the case that ϵ transitions are needed to reach a final state, so $y_i = \epsilon$ is possible). By definition of an NFA-ϵ we have a transition sequence:

  $$(q_0, p_0) \xrightarrow{y_0} (q_1, p_1) \xrightarrow{y_1} \dots \xrightarrow{y_n} (q_f, p_f)$$

  The construction of NFA-ϵ M' tells us that for each such transition, there exists an $x_i \in \Sigma \cup \{\epsilon\}$ such that

  $$p_i x_i \to_T y_i p_{i+1}$$

  The combination of these rules, $p_0 w \to_T^{\#} v p_f$ where $w = x_0 x_1 \dots x_n$. It is left to show that $w \in \mathcal{L}(M)$. Since there can be an i for which $x_i = \epsilon$, we need to show that this does not matter for acceptance by M. Indeed, if $x_i = \epsilon$ then $\delta(q_i, x_i) = q_i$. Hence in the transition sequence, we can see that $q_i = q_{i+1}$. Now w is accepted by M and we can therefore conclude that $v \in f_T(L)$.

- Suppose $w \in \mathcal{L}(M)$ and $f_T(w) = v$ with $w = x_0 x_1 \dots x_n$ and $v = y_0 y_1 \dots y_n$. Then the transduction $p_0 w \to_T^{\#} v p_f$ exists in $S^{\#}$. We decompose rules in $S^{\#}$ into smaller ones via the algorithm 1. We get a sequence of rules:

  $$p_0 \xrightarrow{x_0/y_0} p_1 \xrightarrow{x_1/y_1} \dots \xrightarrow{x_n/y_n} p_f$$

Since by assumption we know that w is accepted by M, there is also a transition sequence:

$$q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} \dots \xrightarrow{x_n} q_f$$

The construction of M' now allows us to create the following transition sequence:

$$(q_0, p_0) \xrightarrow{y_0} (q_1, p_1) \xrightarrow{y_1} \dots \xrightarrow{y_n} (q_f, p_f)$$

Now some letters in w or v can be ϵ. It does not matter since if $x_i = \epsilon$ for an i then in the transition sequence we just have that $(q_i, p_i) \xrightarrow{y_i} (q_i, p_{i+1})$. It is also no problem if $y_i = \epsilon$ for some i because we have an ϵ transition, and that is allowed in an NFA-ϵ. We conclude $v \in \mathcal{L}(M')$. □

3.1.3 Composing transducers

Regular languages are closed under union, intersection, complement, Kleene star, and transductions. Unfortunately, regular languages are not closed under infinite union/intersection, meaning that maybe, infinite transductions also will not preserve regularity. What are infinite transductions? We need compositions for that. Taking a look at figure 3.1, we see that $f_T(\{abbba\}) = baab$. We know that function composition is possible if the range of one function is the same as the domain of the other. However, that is not sufficient for us. One may see a transducer as a function, but functions are generally not transducers. We, therefore, have to construct a new transducer from two existing transducers.

Definition 3.1.6. Let $T = \langle Q, q, \Sigma, \Gamma, R_T, F_T \rangle$ and $S = \langle P, p, \Gamma, \Delta, R_S, F_S \rangle$ be transducers. Then $S \circ T = \langle Q \times P, \langle q_0, p_0 \rangle, \Sigma, \Delta, R_{S\circ T}, F_T \times F_S \rangle$ is a transducer with $f_{S\circ T} : \Sigma^* \to \wp(\Delta^*)$. Here $R_{S\circ T}$ is defined as: for every rule $\langle q_i, x, y, q_j \rangle \in R_T$ and for every rule $\langle p_m, y, z, p_n \rangle \in R_S$ with $|x|, |y|, |z| \leq 1$, we create the rule $\langle \langle q_i, p_m \rangle, x, z, \langle q_j, p_n \rangle \rangle$ to be in $R_{S\circ T}$.

Proposition 3.1.7. *Let $f_T : \Sigma^* \to \wp(\Gamma^*)$ and $f_S : \Gamma^* \to \wp(\Delta^*)$ be transducers. Then $f_{S\circ T} = f_S \circ f_T$.*

Proof. Let $w \in \Sigma^*$ and assume T and S are both in TNF. By definition we have

$$f_{S\circ T}(w) = \{v \mid \text{There exists a rule in } R^{\#}_{S\circ T} \text{ such that } q_0 w \to^{\#} v q_f\}$$

This means that $v \in f_{S\circ T}(w)$ if and only if $\langle \langle q_0, p_0 \rangle, w, v, \langle q_f, p_f \rangle \rangle \in R^{\#}_{S\circ T}$. Apply algorithm 1 to obtain the set of letter-to-letter rules. Call this set V. By construction of $R_{S\circ T}$ any rule $\langle \langle q_i, p_m \rangle, x, z, \langle q_j, p_n \rangle \rangle \in V$ is made from two rules: $\langle q_i, x, y, q_j \rangle$ and $\langle p_m, y, z, p_n \rangle$. We have two sequences.

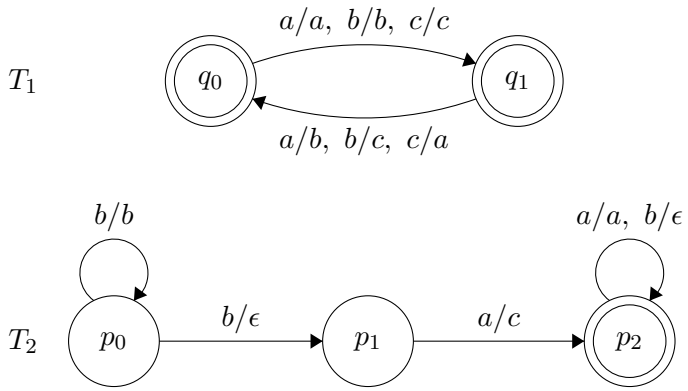

Figure 3.2: Two transducers that we can compose.

One sequence of $\langle q_i, x, y, q_j \rangle \in R_T$ which combines to $\langle q_0, w, u, q_f \rangle$ for some $u \in \Gamma^*$ and another sequence of $\langle p_m, y, z, p_n \rangle \in R_S$ which combines to $\langle p_0, u, v, p_f \rangle$. Therefore:

$$v \in f_{S \circ T}(w) \text{ if and only if } v \in \bigcup_{w' \in \{u \mid q_0 w \to_T^\# u q_f\}} \{v \mid p_0 w' \to_T^\# v q_f\}$$

The set $\{u \mid q_0 w \to_T^\# u q_f\}$ is just $f_T(w)$ and $\{v \mid p_0 w' \to_T^\# v q_f\} = f_S(w')$. We see that $v \in f_{S \circ T}(w)$ if and only if $v \in \bigcup_{w' \in f_T(w)} f_S(w')$. The latter is equal to $f_S \circ f_T(w)$ □

Take a look at figure 3.2. This figure shows two transducers for which we want to find the composition. Figure 3.3 is the result of the composition. We are looking for arrows labeled x/y in T_1 and arrows labeled y/z in T_2. Here $x, y, z \in \{a, b, c\}$. For example, we see in T_1 the rule $q_0 b \to_{T_1} b q_1$ and in T_2 we see the rule $p_0 b \to_{T_1} b p_0$. Thus in $T_2 \circ T_1$ we get the rule $\langle q_0, p_0 \rangle b \to_{T_2 \circ T_1} b \langle q_1, p_0 \rangle$. Another, more interesting, example are the rules $q_1 a \to_{T_1} b q_0$ in T_1 and $p_0 b \to_{T_2} \epsilon p_1$ in T_2. Combining these will yield $\langle q_1, p_0 \rangle a \to_{T_2 \circ T_1} \epsilon \langle q_0, p_1 \rangle$. If we continue this procedure of finding such rules, we will find all the rules, and it will result in the transducers in figure 3.3.

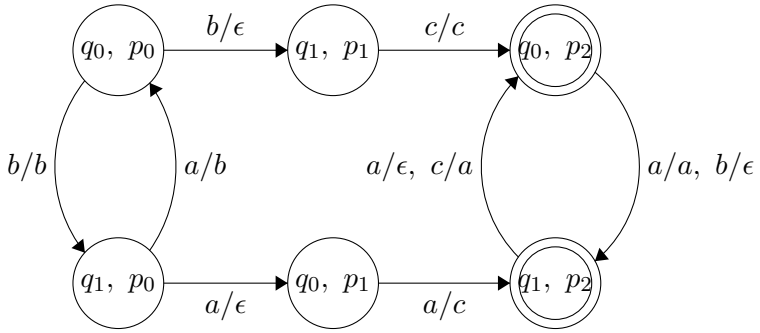

Figure 3.3: The composition of two transducers in figure 3.2

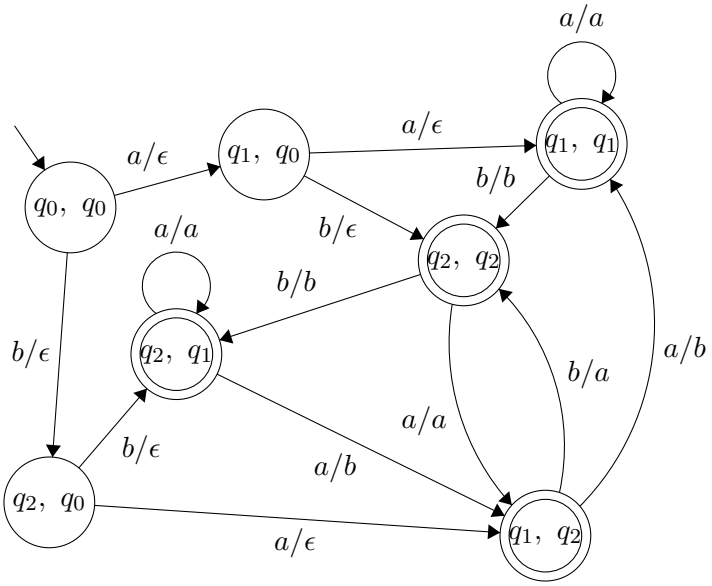

Figure 3.4: T^2 of the transducer in figure 3.1

3.2 ω-iteration

We can go further than only one composition. Suppose we have a sequence of transducers $T_0, T_1, \ldots, T_n$. As long as the output alphabet of T_i is the same as the input alphabet of T_{i+1}, then we can make a composition $f_{T_n} \circ f_{T_{n-1}} \circ f_{T_{n-2}} \circ \cdots \circ f_{T_0}$. This is possible due to induction and proposition 3.1.7. Imagine now that we have a transducer T such that $f_T : \Sigma^* \to \wp(\Sigma^*)$. We can compose f_T with itself twice or even thrice. We refer to this kind of composition as "iteration" since we iterate over the natural numbers (the number of times we apply f_T to the input word).

Definition 3.2.1 (Iteration)**.** Let T be a transducer over the alphabet Σ (meaning that the input and output alphabets of T are both Σ). Then we define iteration of transducers inductively as:

- $T^0 = \langle \{q_0\}, q_0, \Sigma, \Sigma, \{\langle q_0, x, x, q_0\rangle \mid x \in \Sigma\}, \{q_0\}\rangle$
- $T^{n+1} = T \circ T^n$

T^0 is the identity transducer; if we apply f_{T^0} on a language L, the result will be L. We take a look at an example of an iterated transducer. Suppose we have the transducer T from figure 3.1. We apply regular composition as we did with the transducers in figure 3.2 only now our T_1 and T_2 are both T. The composition will result in the transducer in figure 3.4. It is our discretion to iterate many more times if we wish to do so. The resulting composition will also preserve the regularity of a language. This is a corollary of theorem 3.1.5 which we prove by induction on the number of iterations.

Corollary 3.2.1.1. *Let T be a transducer over the alphabet Σ and let L be a regular language. Then we have that $f_{T^n}(L)$ is regular for every $n \in \mathbb{N}$.*

Proof. We prove the statement by induction on n.

- Suppose $n = 0$. Then $f_{T^0}(L) = L$ which is regular by assumption.
- Suppose $f_{T^n}(L)$ is regular.

  Consider $f_{T^{n+1}}(L) = (f_T \circ f_{T^n})(L) = f_T(f_{T^n}(L))$. By our induction hypothesis, $f_{T^n}(L)$ is regular and by theorem 3.1.5, $f_{T^{n+1}}(L)$ is regular.

Hence, we may conclude $f_{T^n}(L)$ is regular for every $n \in \mathbb{N}$. □

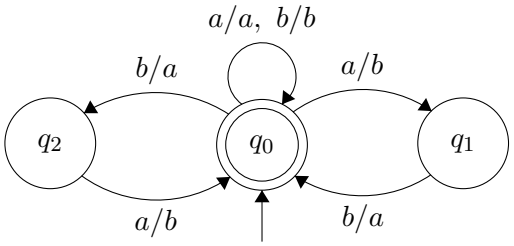

Figure 3.5: A FST that can change the substring ab to ba and vice versa. The transducer does not have to change the substrings, it can also leave them as they are.

Regular languages are closed under finite intersection, union, and more. However, we know that taking the infinite intersection or union of regular languages is not regular. This is also the case with infinite transductions [8]. To prove that regular languages are not closed under infinite transductions, we consider the ω-iteration[1]:

Definition 3.2.2. Let T be a transducer over the alphabet Σ and let L be a language. The ω-iteration of T is defined as:

$$f_{T^{<\omega}}(L) = \bigcup_{i \in \mathbb{N}} f_{T^i}(L)$$

We are taking the union of all iterated transducers. That is $f_{T^{<\omega}}(L) = L \cup f_T(L) \cup f_{T^2}(L) \cup \ldots$ An analogue definition can be made without the language L. One would then have:

$$f_{T^{<\omega}} = \bigcup_{i \in \mathbb{N}} f_{T^i}$$

The transducer $T^{<\omega}$ is in itself an infinite state transducer. Nevertheless, infinite-state transducers are hard to work with or even conceptualize. Although this transducer itself is not in the scope of this thesis, Dennis Dams et al. created an algorithm to reduce the size of $T^{<\omega}$ to an equivalent transducer [4]. However, this reduced transducer does not have to be finite.

[1]The symbol ω comes from ordinal numbers, where ω is the first inductive ordinal number. The ordinal ω is also the smallest infinity.

3.2.1 Non-regularity of ω-iteration

It is, unfortunately, the case that ω-iteration does not preserve regularity. This negative result is shown in [8], but an explicit counterexample is not given. For the remainder of this section, we will show a transducer that will be our counterexample and prove that ω-iteration does not preserve regularity.

The transducer in figure 3.5 is a counterexample. We will call this T from here on forward. The transducer T switches a with b and vice versa if a and b are adjacent to each other (ab or ba). However, it can also leave a string as it is. For instance, take the word $abba$. Feeding this to T will yield $f_T(abba) = \{abba, baba, abab, baab\}$. It shows that the transducer is non-deterministic. This is not a problem for iteration since $f_{T^2}(abba) = \bigcup_{w \in f_T(abba)} f_T(w) = f_T(abba) \cup f_T(baba) \cup f_T(abab) \cup f_T(baab)$. We will prove that ω-iteration does not preserve regularity in general.

Theorem 3.2.3. *Let T be the transducer in fig 3.5. Let $\Sigma = \{a, b\}$ and let $L = \{(ab)^n \mid n \in \mathbb{N}\}$. Then we have:*

$$f_{T^{<\omega}}(L) = \{w \in \Sigma^* \mid |w|_a = |w|_b\}$$

We first need some lemmas before we can start to prove the theorem. It is important to know that our transducer of choice T preserves the property $|w|_a = |w|_b$.

Lemma 3.2.4.

$$\forall_{i \in \mathbb{N}}[f_{T^i}(L) \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}]$$

Proof. We prove this by induction on i.

- Suppose $i = 0$.

  Then $f_{T^0}(L) = L = \{(ab)^n \mid n \in \mathbb{N}\} \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}$.

- Suppose that the statement holds for i. We now are proving the statement for $i + 1$. We assume $w \in f_T(f_{T^i}(L))$. Then we know that there is a $v \in f_{T^i}(L)$ such that $w \in f_T(v)$. By the induction hypothesis, we can conclude that $|v|_a = |v|_b$. Now we apply f_T on v and show that for all $u \in f_T(v)$, $|u|_a = |u|_b$. The transducer T has a total of six rules:

  $$\begin{array}{lll} q_0a \to aq_0 & q_0a \to bq_1 & q_1b \to aq_0 \\ q_0b \to bq_0 & q_0b \to aq_2 & q_2a \to bq_0 \end{array}$$

  Application of the rules $q_0a \to aq_0$ and $q_0b \to bq_0$ will not change the number of symbols. Using the rule $q_0a \to bq_1$ gives an inequality $|u|_a < |u|_b$ but after this rule, we are in a non-accepting state.

So any transduction that ends in q_1, is rejected. The only rule to get out of q_1 is $q_1 b \to a q_0$. This rule acts as an inverse because it changes a b into an a. We get $|u|_a = |u|_b$ after applying these two rules. A similar reasoning holds for the rules $q_0 b \to a q_2$ and $q_2 a \to b q_0$. Therefore, $|u|_a = |u|_b$ for any $u \in f_T(v)$ and especially for w.

Now we have that $f_{T^i}(L) \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}$. □

For the proof of theorem 3.2.3, it will be necessary that we can translate any word of length $2n$ to $a^n b^n$ and back. Proving that this is the case for T, we need inversions. An inversion is an occurrence of the letter a to the right of a letter b. The word ba contains one inversion, baa and bba have two inversions. Also $a^n b^n$ has zero inversions for any n.

In the next lemma, we prove that the transduction f_T has the property that if a word w of length $2n$ is in $\{w \in \Sigma^* \mid |w|_a = |w|_b\}$ then $a^n b^n \in f_T(w)$. We prove this by induction on the number of inversions. Swapping an a with a b either increases or decreases the number of inversions by one. That insight is what we will be using.

Lemma 3.2.5. *From any word $w \in \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ of length $2n$, we can obtain the word $a^n b^n$ by applying f_T to w in a finite number of steps.*

Proof. We prove the lemma by induction on the number of inversions $p \in \mathbb{N}$.

- Suppose $p = 0$. Then $w = a^n b^n$. We therefore need zero applications of f_T.
- Suppose $w' \to_T^k a^n b^n$ for a $k \in \mathbb{N}$ when w' has p inversions. We prove that if w has $p + 1$ inversions, then $w \to_T^{k+1} a^n b^n$. Since w has $p + 1 > 0$ inversions, w contains a substring ba. Apply f_T to w such that $ba \to_T ab$ for this substring. Call this transduced word w'. Now w' has p inversions. By the induction hypothesis, we have that $w' \to_T^k a^n b^n$. Thus we have $w \to_T w' \to^k a^n b^n$ and therefore $w \to_T^{k+1} a^n b^n$. □

At last, we need a similar yet different lemma for us to prove theorem 3.2.3.

Lemma 3.2.6. *Any word $w \in \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ of length $2n$ can be obtained by applying f_T to $a^n b^n$ finitely many times.*

Proof. We prove this again by induction on the number of inversions $p \in \mathbb{N}$.

- Suppose $p = 0$, then $w = a^n b^n$ so no applications of f_T are needed on $a^n b^n$.

- Suppose that if a word w' has p inversions, then we can translate $a^n b^n$ to w' in $k < \infty$ steps. Suppose w has $p+1$ inversions. Then it must have a substring ba. Now swap this instance of ba to ab and call this new word w'.

  The word w' now has p inversions, and so by the induction hypothesis, we have that $a^n b^n \to_T^k w'$. Now we apply f_T to w' such that the substring ab becomes ba again, thus we have w. Now we have $a^n b^n \to_T^{k+1} w$.

We conclude that with finitely many applications of f_T on $a^n b^n$, we get any word $w \in \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ of length $2n$. □

It is time to prove theorem 3.2.3.

Proof. We will prove separately that $f_T^{<\omega}(L) \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ and $\{w \in \Sigma^* \mid |w|_a = |w|_b\} \subseteq f_{T^{<\omega}}(L)$.

- $f_T^{<\omega}(L) \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}$. Assume $w \in f_T^{<\omega}(L)$. Then we have that $\exists_{i \in \mathbb{N}}[w \in f_{T^i}(L)]$. Furthermore, by lemma 3.2.4 we know that it does not matter how many times we apply f_T to w. Thus $f_T^i(w) = f_{T^i}(w) \in \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ for any $i \in \mathbb{N}$. Therefore, we are able to conclude:

$$f_{T^{<\omega}}(L) \subseteq \{w \in \Sigma^* \mid |w|_a = |w|_b\}$$

- $\{w \in \Sigma^* \mid |w|_a = |w|_b\} \subseteq f_{T^{<\omega}}(L)$.

  Suppose $w \in \{w \in \Sigma^* \mid |w|_a = |w|_b\}$ and suppose that $|w| = 2n$ for a $n \in \mathbb{N}$. Consider the word $a^n b^n$. Then w has a finite number p of inversions. Here we can apply lemma 3.2.5 on $(ab)^n$ to get $a^n b^n$ in k_0 steps. By lemma 3.2.6 we can transduce $a^n b^n$ to w in k_1 applications of f_T. We then have:

$$(ab)^n \to_T^{k_0} a^n b^n \to_T^{k_1} w \Rightarrow (ab)^n \to_T^{k_0+k_1} w$$

  By definition of $f_{T^{<\omega}}$ and the fact that $w \in f_T^{k_0+k_1}(L) = f_{T^{k_0+k_1}}(L)$ we conclude that $w \in f_{T^{<\omega}}(L)$ which was to be proven. □

Corollary 3.2.6.1. *ω-iteration does not preserve regularity in general.*

Proof. The language $\{w \in \Sigma^* \mid |w|_a = |w|_b\}$ is not regular. □

Chapter 4

H-closures

In 1995, E. Cohen introduced hypotheses in Kleene Algebras, from where the H-closure originated [3]. We give a proper definition of the H-closure in this chapter. From the definition, we introduce a proof system used in [5] to determine the inclusion of languages in the H-closure. Furthermore, we give a different step-by-step method for determining the H-closure.

4.1 What are H-closures?

It is not always possible to determine program equivalence. Take, for instance, this example from [5]. We have two regular expressions $(a^*b)^*$ and $((a+b)^*b+1)$. It is decidable if regular expressions are equivalent. We can check if these regular expressions are accepted by the same minimized DFA (up to isomorphism) [7]. In the case of $(a^*b)^*$ and $((a+b)^*b+1)$, these are equivalent.

Consider another example from [5], $(a+b)^*$ and b^*a^*. These expressions are not equivalent since $ab \in [\![(a+b)^*]\!]$ but $ab \notin [\![b^*a^*]\!]$. But what if we know that $ab = b$? Then we would have $ab = b \in [\![b^*a^*]\!]$. With this hypothesis, we can determine that $(a+b)^*$ and b^*a^* are equivalent. We can change every subword ab in some word w in $[\![(a+b)^*]\!]$ to b and then w will be in $[\![b^*a^*]\!]$.

The idea of using hypotheses needs to be more rigorous in order to prove that the hypothesis "$ab = b$" is sufficient for $(a+b)^*$ and b^*a^* to be equivalent. First some notation. If e, f are regular expressions, then $e \leq f$ means f "translates to" e and this is called an hypothesis. Note here that this is only an informal definition of $\leq$, sufficient for our purposes. When it is the case that $e \leq f$ and $f \leq e$, we can say $e = f$.

Considering the example again, $ab = b$ is the same as saying $ab \leq b$ and $b \leq ab$. Notice here that it is sufficient to only assume $ab \leq b$ since $[\![b^*a^*]\!] \subseteq [\![(a+b)^*]\!]$. However, just assuming $ab \leq b$ does not magically allow us to say $[\![b^*a^*]\!] = [\![(a+b)^*]\!]$. We need to apply our hypotheses on $[\![b^*a^*]\!]$. Doing so will extend $[\![b^*a^*]\!]$ to a language say C where $[\![b^*a^*]\!] \subseteq C$ contains words w that after a number (possibly infinite) of applications of $ab \leq b$ will be in $[\![b^*a^*]\!]$. This set C is called the H-closure.

Definition 4.1.1 (H-closure [9])**.** Let H be a set of hypotheses and $L \subseteq \Sigma^*$ a language. The H-closure of L, written as $\mathrm{cl}_H(L)$, is the smallest language such that:

- $L \subseteq \mathrm{cl}_H(L)$
- For all $e \leq f \in H$, for each $u, v \in \Sigma^*$, if $u[\![f]\!]v \subseteq \mathrm{cl}_H(L)$ then $u[\![e]\!]v \subseteq \mathrm{cl}_H(L)$

The words u and v are contexts that allow us to use our hypotheses in the middle of the word instead of requiring that we translate the entire word. Language equivalence with hypotheses is determined with the H-closure. Regular expressions e and f are equivalent if and only if $\mathrm{cl}_H([\![e]\!]) = \mathrm{cl}_H([\![f]\!])$.

4.2 A helpful proof system

Determining $\mathrm{cl}_H([\![b^*a^*]\!])$ is more challenging but not undoable. First note that $\mathrm{cl}_H([\![b^*a^*]\!]) \subseteq [\![(a+b)^*]\!]$ since $[\![(a+b)^*]\!]$ contains all the words using the alphabet $\{a, b\}$. For the second part, we want $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$ meaning that for any word $w \in [\![(a+b)^*]\!]$, we want a word $v \in [\![b^*a^*]\!]$ such that we can get from v to w using $ab \leq b$. This can be solved intuitively by starting with $v = b^n$ where $n = |w| - |w|_a$. We can then use the hypothesis exactly at the b's in v when there is an a in front of that b in w.

For instance, let $w = ababb$. Then $v = bbb$. In order to get to w from v, we use the hypothesis $ab \leq b$ on the first and second b in v because in w, the first and second b have an a in front. One might think: what about words such as $aaabb$ or $aaaaa$? In the first case, we want to use the hypothesis three times on the first b. The second case is solved by defining v as $aaaaa$. Via this intuitive reasoning, we can determine that $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$.

However, this is not a solid proof. We reasoned why this must be true. Proving $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$ using the definition is possible but not preferred. We are better off using a proof system for this task.

Definition 4.2.1 ([5])**.** Let H be a set of hypotheses and L a regular language. We define a proof system with the following three rules:

$$\frac{}{u}\, u \in L \text{ (1)}, \quad \frac{(u)_{u\in[\![e]\!]}}{e} \text{ (2)}, \quad \frac{ufv}{uwv}\, w \in [\![e]\!], e \leq f \in H \text{ (3)}$$

If e is derivable in this proof system, we denote that by $\vdash_{H,L} e$. Such a tree is called a derivation tree for $[\![e]\!] \subseteq \mathrm{cl}_H(L)$.

The proof system will help us prove $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$.

Proof. We use induction on the structure of the derivation tree. We start our derivation tree as follows:

$$\frac{(u)_{u\in[\![(a+b)^*]\!]}}{(a+b)^*} \text{ (2)}$$

There are two cases we need to cover. Since u is a word, (2) will not provide any new information.

1. Suppose we use (1). We see $u \in [\![b^*a^*]\!]$ and thus we are done.
2. Suppose we use (3). Let u be of the form $xaby$ with $x, y \in \{a, b\}^*$ and of length n. Assume that any word in $[\![(a+b)^*]\!]$ of length $k < n$ we have a derivation tree. Since we have $xaby$ and $ab \leq b$, we obtain the proof tree:

$$\frac{\dfrac{\overline{xby}}{xaby_{xaby\in[\![(a+b)^*]\!]}}\, ab \leq b \text{ (3)}}{(a+b)^*} \text{ (2)}$$

   Notice that $|xby| < |xaby|$ and thus by our assumption, we have a derivation tree for xby. Hence a proof tree of length n asserts $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$.

By induction we see that $[\![(a+b)^*]\!] \subseteq \mathrm{cl}_H([\![b^*a^*]\!])$. □

With the proof system, we were able to rigorously prove

$$[\![(a+b)^*]\!] = \mathrm{cl}_{\{ab\leq b\}}([\![b^*a^*]\!])$$

Yet we must show that such derivation trees actually do guarantee that the language of an expression is contained in the closure.

Proposition 4.2.2 (App. A [5])**.**

$$[\![e]\!] \subseteq cl_H(L) \leftrightarrow \vdash_{H,L} e$$

4.3 Computation of the closure

Besides the proof system, there is another method for determining such closures, and this method is somewhat related to transducers. Consider the language $L = \{b\}$ and let the set of hypotheses be $H = \{a \leq bb^*, bb \leq b\}$. We want to determine the H-closure of L, and we do this step by step. We are applying one hypothesis at a time. At first, we can only use $bb \leq b$ on b. We get a new word bb after the first step. The second step is to use our hypotheses on bb. There is again only one option, $bb \leq b$. Thus we get one new word bbb after the second step. We keep taking steps an infinite number of times. Each step results in a string of b's with an extra b concatenated to the end. After a countable number of steps, we group the words and have the set $\{b, bb, bbb, bbbb, bbbbb, \dots\}$. This set is not the closure yet! After a countable number of steps, we have the set $[\![bb^*]\!]$. We can translate this whole set to a by using $a \leq bb^*$. Thus $\mathrm{cl}_H(L) = [\![a + bb^*]\!]$ since there are no hypotheses that can be used on a.

Mathematically, the procedure of taking steps is a form of induction called transfinite induction[1]. In the example, we take a countable number of steps and then one more step (hence the name transfinite). One such step can define a new word. In mathematical terms:

$$st_H(L) = \bigcup \{u[\![e]\!]v \mid e \leq f \in H, u, v \in \Sigma^*, u[\![f]\!]v \subseteq L\}$$

So in our example with $L = \{b\}$ and $H = \{a \leq bb^*, bb \leq b\}$, we would have:

$$\begin{aligned} st_H(\{b\}) &= \{ubbv \mid u, v \in \Sigma^*, ubv \in \{b\}\} \cup \{uav \mid u, v \in \Sigma^*, u[\![bb^*]\!]v \subseteq \{b\}\} \\ &= \{bb\} \cup \varnothing \\ &= \{bb\} \end{aligned}$$

Any next step, $st_H^{\alpha+1}(L)$, is calculated as taking a step from the previous result, $st_H(st_H^{\alpha}(L))$:

$$\begin{aligned} &st_H(st_H(\{b\})) = st_H(\{bb\}) \\ &= \{ubbv \mid u, v \in \Sigma^*, ubv \in st_H(\{b\})\} \cup \{uav \mid u, v \in \Sigma^*, u[\![bb^*]\!]v \subseteq st_H(\{b\})\} \\ &= \{ubbv \mid u, v \in \Sigma^*, ubv \in \{bb\}\} \cup \{uav \mid u, v \in \Sigma^*, u[\![bb^*]\!]v \subseteq \{bb\}\} \\ &= \{bbb\} \end{aligned}$$

[1] Transfinite induction is induction on ordinal numbers.

Any step after an infinite amount of steps ($st_H^\lambda(L)$) is calculated by grouping all the previous steps together ($\bigcup_{\alpha<\lambda} st_H^\alpha(L)$):

$$\begin{aligned} st_H^\omega(L) &= \bigcup_{\alpha<\omega} st_H^\alpha(L) \\ &= \{b\} \cup \{bb\} \cup \{bbb\} \cup \dots \\ &= \{b, bb, bbb, \dots\} \\ &= [\![bb^*]\!] \end{aligned}$$

All these steps derive new words, and when we do all the steps (that is, we do not find new words after taking a step), we have all the words in the closure. In other words:

$$\mathrm{cl}_H(L) = \bigcup_{Ord(\alpha)} st_H^\alpha(L)$$

Definition 4.3.1 ([5])**.** The step-by-step method is defined inductively in the following way.

- The first step: $st^0(L) = L$.
- The successor step: $st_H^{\alpha+1}(L) = st_H(st_H^\alpha(L))$.
- The limit step: if λ is a limit ordinal, then $st_H^\lambda(L) = \bigcup_{\alpha<\lambda} st_H^\alpha(L)$.

A step is defined as:

$$st_H(L) = \bigcup \{u[\![e]\!]v \mid e \leq f \in H, u, v \in \Sigma^*, u[\![f]\!]v \subseteq L\}$$

The step-by-step method is one effective way of determining closures. It is also the method that resembles how transducers work.

Chapter 5

Relation between transducers and H-closures

Up until now, we have discussed both transducers and H-closures. Both definitions were discussed in chapters 3 and 4 respectively. This chapter presents and proves our two new theorems that relate to the two notions. H-closures and iterated transducers are functions on languages, and in specific circumstances, they are the same function.

5.1 Hypotheses and transducers

Let us take a look at figure 5.1. The transducer T in this figure can translate any instance of the subword ba to ab. We can try some language and see what happens. Suppose $L = \{bbaa\}$. Then $f_T(L) = \{bbaa, baba\}$. We will see something particular happening if we try to iterate T a few times.

$$f_T(L) = \{bbaa, baba\}$$
$$f_{T^2}(L) = \{bbaa, baba, abba, baab, abab\}$$
$$f_{T^3}(L) = \{bbaa, baba, abba, baab, abab, aabb\}$$

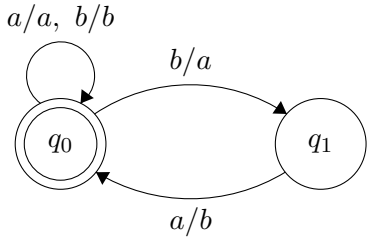

Figure 5.1: A transducer that may translate ba to ab.

Any further iterations of T will not lead to more elements (there are only six permutations of $bbaa$). Therefore $f_{T^{<\omega}}(L) = \{bbaa, baba, abba, baab, abab, aabb\}$. With this in mind, let $H = \{ab \leq ba\}$ and consider $\mathrm{cl}_H(L)$. From chapter 4, we know that $\mathrm{cl}_H(L) = \bigcup_\alpha \mathrm{st}_H^\alpha(L)$. Some calculations lead to the following:

$$\begin{aligned}
\mathrm{st}_H(L) &= \{baba\} \\
\mathrm{st}_H^2(L) &= \{abba, baab\} \\
\mathrm{st}_H^3(L) &= \{abab\} \\
\mathrm{st}_H^4(L) &= \{aabb\}
\end{aligned}$$

For $\alpha > 4$, $\mathrm{st}_H^\alpha(L) = \varnothing$. We can therefore see that

$$\bigcup_{Ord(\alpha)} \mathrm{st}_H^\alpha(L) = \{bbaa, baba, abba, baab, abab, aabb\} = \mathrm{cl}_H(L)$$

One may notice that it happens to be the case that

$$\mathrm{cl}_H(L) = \{bbaa, baba, abba, baab, abab, aabb\} = f_{T^{<\omega}}(L)$$

Even more surprising, the phenomenon is not a coincidence. For specific hypotheses and transducers, such equalities arise. We can find two types of such equalities, one given we have a transducer and another given a set of hypotheses.

5.2 From a transducer to hypotheses

The first theorem shows that given a transducer T, we can modify T to T_c such that we can create a set of hypotheses of the form $f_{T_c}(w) \leq w$ with the property that the ω-iteration yields the H-closure. A transducer T is not sufficient since T may not be able to read the context. Take for example $bbababaa$ with $ab \leq ba$ as an hypothesis. We want to apply the hypothesis in the middle of the word, $bba\mathbf{ba}baa$. The transducer T is thus supposed to let the words bba and baa intact. Therefore we need a new transducer T_c that can do that. The construction of T_c is relatively simple: we add two states, a start, and an accepting state, that both read context. The start state reads letters and will transition into the T where the hypotheses are applied. After that is done, we transition to the accepting state and read the rest of the word. The claim is that the ω-iteration of T_c will be the same as the H-closure with $H = \{f_{T_c}(w) \leq w \mid w \in \Sigma^*\}$.

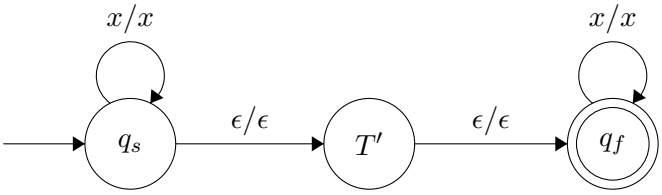

Figure 5.2: The form of transducer we need for the proof of theorem 5.2.1

Theorem 5.2.1. *Let $T = \langle Q, q_0, \Sigma, \Sigma, R, F\rangle$ be a transducer over the alphabet Σ. Then there exists a transducer T_c such that $H = \{f_{T_c}(w) \leq w \mid w \in \Sigma^*\}$, L a language and $cl_H(L) = f_{T_c^{<\omega}}(L)$.*

Proof. We define $T_c = \langle Q_c, q_s, \Sigma, \Sigma, R_c, F_c\rangle$ where $Q_c = Q \cup \{q_s, q_f\}$, $F_c = \{q_f\}$ and R_c as the union of R, $\{\langle q_s, \epsilon, \epsilon, q_0\rangle\}$, $\{\langle q_t, \epsilon, \epsilon, q_f\rangle \mid q_t \in F\}$, $\{\langle q_s, x, x, q_s\rangle \mid x \in \Sigma\}$ and $\{\langle q_f, x, x, q_f\rangle \mid x \in \Sigma\}$. The states q_s and q_f are the initial and final states of T_c respectively and are not part of the original transducer T. The transition relation R_c is a union of the transition relation R and additional rules that preserve context. E.g., the new rules translate letters to themselves. The two ε-to-ε rules ensure we transition in and out of the transducer T. See figure 5.2 for an illustration. This T_c now has the property $uf_T(w)v \subseteq uf_{T_c}(w)v \subseteq f_{T_c}(uwv)$ for every $u, v \in \Sigma^*$ and $w \in L$. We can see that since if a word $w_1 \in uf_T(w)v$ then it has the form uw_2v with $w_2 \in f_T(w)$. The word w_2 therefore is a translation. The adjusted transducer T_c translates a word w_2 the same as T would do. Therefore $uf_T(w)v \subseteq uf_{T_c}(w)v$. The transducer T_c is designed have the ability to preserve context. This means that $f_{T_c}(uwv)$ must contain $uf_{T_c}(w)v$.

- We prove $f_{T_c^{<\omega}}(L) \subseteq \mathrm{cl}_H(L)$. Assume that $w \in f_{T_c^{<\omega}}(L)$. Then definition 3.2.2 tells us there is an $i \in \mathbb{N}$ such that $w \in f_{T_c^i}(L)$. We perform induction on the number i of applications of f_{T_c}. Suppose $i = 0$. Then $w \in f_{T^0}(L) = L$ and therefore we can use (1) according to definition 4.2.1. We conclude $w \in \mathrm{cl}_H(L)$. Now we assume that if $v \in f_{T_c^i}(L)$ then $v \in \mathrm{cl}_H(L)$ (IH). Suppose $w \in f_{T_c^{i+1}}(L)$. By definition, we know there must be a $w' \in L$ such that $w \in f_{T_c^{i+1}}(w')$. Then using some rewriting we conclude the following:

$$f_{T_c^{i+1}}(w') = f_{T_c}(f_{T_c^i}(w')) = \bigcup_{v \in f_{T_c^i}(w')} f_{T_c}(v)$$

  Therefore there is some $v \in f_{T_c}(w')$ such that $w \in f_{T_c}(v)$. We can now construct a derivation tree.

$$\dfrac{\dfrac{\begin{matrix}-\\ v\end{matrix}\ \text{(IH)}}{(w)_{w \in f_{T_c}(v)}}\ (3),\ f_{T_c}(v) \leq v \in H}{f_{T_c^{<\omega}}(L)}\ (2)$$

By induction, we have constructed a finite proof tree and we may conclude $f_{T_c^{<\omega}}(L) \subseteq \mathrm{cl}_H(L)$.

- Let $uwv \in f_{T_c^{<\omega}}(L)$ such that $w \in f_{T_c^j}(w')$ for some $w' \in L$ and $j \in \mathbb{N}$. The goal is to establish that $uf_{T_c}(w)v \subseteq f_{T_c^{<\omega}}(L)$. When this holds, we know by definition of H-closure that $\mathrm{cl}_H(L) \subseteq f_{T_c^{<\omega}}(L)$. Applying f_{T_c} gives us $uf_{T_c}(w)v \subseteq uf_{T_c}(f_{T_c{}^j}(w'))v \subseteq f_{T_c}(uf_{T_c^j}(w')v)$. So we can obtain $uf_{T_c}(w)v$ from earlier translations. We obtain $uf_{T_c}(w)v \subseteq f_{T_c^{<\omega}}(L)$. Thus we have $u\{w\}v \subseteq f_{T_c^{<\omega}}(L)$ implies $uf_{T_c}(w)v \subseteq f_{T_c^{<\omega}}(L)$. Since $\mathrm{cl}_H(L)$ is the smallest set for which such property holds, we conclude $\mathrm{cl}_H(L) \subseteq f_{T_c^{<\omega}}(L)$. □

5.3 From hypotheses to a transducer

The second theorem arises when we start with hypotheses instead of transducers. These hypotheses need to be of a particular kind. Due to the nature of transducers, we can only consider hypotheses of the form $e \leq w$ where e is an expression and w a word. Was there an inequation of the form $e \leq f$ with f a regular expression, then a transducer would need to read all the words in $[\![f]\!]$. To arrive at the theorem we will discuss later in this section, it is vital to discuss the construction of a transducer-based on a set of hypotheses.

Definition 5.3.1. Let $H = \{e_i \leq w \mid w \in \Sigma^* \wedge i \in \mathbb{N}\}$ with e_i a regular expression. A transducer constructed by H is a transducer built using the algorithm below:

1. Start with a state q_0.
2. For each hypothesis $e_i \leq w$, create the rule $\langle q_0, w, e_i, q_0\rangle$. After this for-loop, we have a transducer T'.
3. Add states q_s, the initial state, and q_f, the accepting state.
4. Add rules $\langle q_s, \varepsilon, \varepsilon, q_0\rangle, \langle q_0, \varepsilon, \varepsilon, q_f\rangle$.
5. For each $x \in \Sigma$, add the rules, $\langle q_s, x, x, q_s\rangle$ and $\langle q_f, x, x, q_f\rangle$.
6. We now have constructed our desired transducer T.

Figure 5.3 shows what such a constructed transducer resembles. The main point of this transducer is that every use of hypothesis is some sequence of transition within T', just like in 5.1 with the hypothesis $\{ab \leq ba\}$. Furthermore, the states q_s and q_f are context readers. As seen with the H-closures, we use context to apply hypotheses within a word. Such context readers therefore allow *bbbaaa* to be translated to *bbabaa*. As an illustration, see figure 5.4.

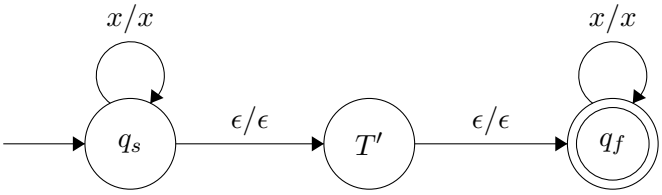

Figure 5.3: General form of a transducer constructed via definition 5.3.1

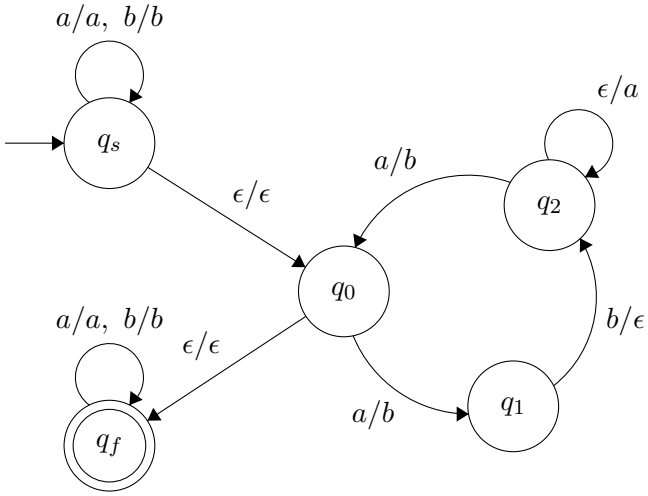

Figure 5.4: A constructed transducer from the hypothesis $ba^*b \leq aba$.

Here T' consists of all the states and transitions in the circle. Again here, the states q_s and q_f are context readers. We first read some or possibly no letters in state q_s.
Then, after applying the hypothesis $ba^*b \leq aba$ zero or more times, we read the remaining letters in state q_f.

The claim is that iterating this constructed transducer an infinite number of times will coincide with the H-closure over the same hypotheses.

Theorem 5.3.2. *Let $H = \{e_i \leq w \mid w \in \Sigma^* \wedge i \in \mathbb{N}\}$ with e_i a regular expression. Let T be a transducer over the alphabet Σ constructed by H and L a language. Then $f_{T^{<\omega}}(L) = cl_H(L)$.*

Proof. We prove separately $f_{T^{<\omega}}(L) \subseteq \mathrm{cl}_H(L)$ and $\mathrm{cl}_H(L) \subseteq f_{T^{<\omega}}(L)$. When we esthablished this, we can conclude that $f_{T^{<\omega}}(L) = \mathrm{cl}_H(L)$

- The goal is to show $f_{T^{<\omega}}(L) \subseteq \mathrm{cl}_H(L)$. Assume $w \in f_{T^{<\omega}}(L)$. When using the same reasoning as in the previous proof, we know that $w \in f_{T^i}(w')$ for some $i \in \mathbb{N}$ and $w' \in L$. We perform induction on the number i of applications of f_T. Suppose $i = 0$. We can now apply (1) from the proof system, and thus we conclude $w \in \mathrm{cl}_H(L)$. Now suppose that if $v \in f_{T^i}(w')$ for an w' in L, then $v \in \mathrm{cl}_H(L)$.

Assume $w \in f_{T^{i+1}(L)}$ meaning that $w \in f_T(v)$ for some $v \in f_{T^i}(w')$.

Important to note that $f_T(u) \leq u \in H$ because $f_T(u)$ is regular and due to the construction given in defintion 5.3.1. We construct a derivation tree with root $f_{T^{<\omega}}(L)$.

$$\dfrac{\dfrac{\overset{-}{v}\ \text{(IH)}}{(w)_{w \in f_T(v)}}\ (3),\ f_T(v) \leq v \in H}{f_{T^{<\omega}}(L)}\ (2)$$

By induction we can conclude $f_{T^{<\omega}}(L) \subseteq \mathrm{cl}_H(L)$.

- We want to prove $uwv \in f_{T^{<\omega}}(L)$ implies $u[\![e_i]\!]v \subseteq f_{T^{<\omega}}(L)$ for any $e_i \leq w \in H$. Let $e_i \leq w \in H$ be given. Suppose $u, v \in \Sigma^*$ and assume $uwv \in f_{T^{<\omega}}(L)$. The assumption gives us an $w' \in L$ and an j such that $uwv \in uf_{T^j}(w')v$. Note that the transducer can opt to write everything it reads; hence $uf_{T^j}(w')v \subseteq f_{T^j}(uw'v)$ and thus context u, v can be preserved. Furthermore, we can apply f_T to uwv to get $f_T(uwv)$. We may conclude $uf_T(w)v \subseteq f_T(uwv)$. Now we can use the construction of T: $u[\![e_i]\!]v \subseteq uf_T(w)v$. This will allow us to finalize the proof. We can apply f_T on $uw'v$ to get $u[\![e_i]\!]v$ in $j+1$ iterations. This means $u[\![e_i]\!]v \subseteq f_{T^{<\omega}}(L)$. We have now shown that $uwv \in f_{T^{<\omega}}(L)$ implies $u[\![e_i]\!]v \subseteq f_{T^{<\omega}}(L)$ for any $e_i \leq w \in H$. Now, as a reminder, $\mathrm{cl}_H(L)$ is the smallest set for which this implication hold. We conclude $\mathrm{cl}_H(L) \subseteq f_{T^{<\omega}}(L)$. □

5.4 A comparison between transducers and H-closures

We established the theorems, and it is compelling to look at the consequences. In several situations, we can choose which method we want to use to determine the closure of some hypotheses; transducers or H-closures. There can be said something for both methods. Each of them has its shortcomings which we will explore.

5.4.1 H-closures are more flexible

We said it multiple times, but transducers cannot read entire languages in one go. For any set of hypotheses H if there is at least one inequation of the from $e \leq f$ with f a regular language that is not a single word, we cannot use transducers to determine the closure. To put it more concretely, suppose we have a hypothesis $b \leq a^*$.

In order to translate to a b, a supposed transducer needs to read $\{\varepsilon, a, aa, aaa, \dots\}$ in one go. One can see that it is not possible. Even the transfinite induction method needs more than a countable amount of steps before it can translate a^* to b. Nevertheless, using the proof system or the step-by-step method can yield an answer if one is looking for what the H-closure might be.

Another reason one might avoid using transducers is because the ω-iteration may be an infinite state machine for a transducer. Using transducers will not be helpful since we cannot easily reason with the ω-iteration. Luckily, this can partially be solved. Efforts have been made to decrease the size of infinite-state ω-iteration [4]. Given the right circumstances, that is, finding a suitable pair of equivalence relations[1] F, P on T^ω, the quotient $T^\omega/F;P$ will preserve the transition relation. So even with an infinite state machine, we can still tell what the closure will be. Nonetheless, finding this pair of future and past bisimulations is a quest on its own, and it is not always possible to find such a pair.

5.4.2 Transducers are more intuitive

With transducers, it is sometimes easier to see what the H-closure is. While H-closures are more potent than transducers, they require intensive reasoning and proving using the proof system or transfinite induction. Transducers, however, allow for some conceptualization. The hypothesis may be complex; a transducer can make it easier to grasp what the inequations mean.

For example, consider the transducer that increments binary numbers. We denote this by T_{bin}. This transducer T_{bin} is depicted in figure 5.5. This transducer takes as an input a binary number x in little-endian (least significant bit first) and returns $x+1$ in little-endian. Such an example could be $37_{10} = 100101_2$. The reverse is 101001 and we apply the composed rule $q_0 101001 \to^{\#}_{T_{bin}} 011001 q_1$. We see $100110_2 = 38_{10}$. Binary increment is not a hard to understand subject. However, we want to see what the closure is, starting from a predetermined binary number. First, what should H be? One can see that we have $1 \leq \varepsilon, 1 \leq 0, 01 \leq 10, 001 \leq 110, \dots, 01 \leq 1, 001 \leq 11, \dots$. Thus we get a rule whenever we have a carry, we add 1 to 0 or when we "overflow".
Hence $H = \{0^n 1 \leq 1^n \mid n \in \mathbb{N}\} \cup \{0^n 1 \leq 1^n 0 \mid n \in \mathbb{N}\} \cup \{1 \leq 0\}$.

[1] "Suitable" here means a swapping pair F, P of future and past bisimulations. These are all defined in [4, Section 2.2]

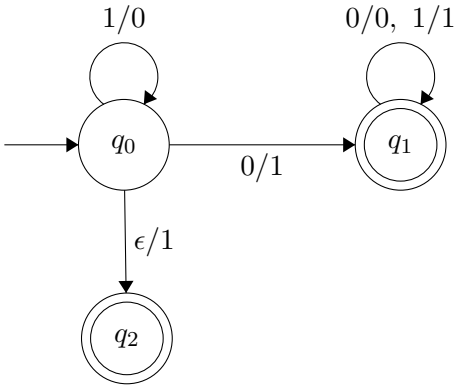

Figure 5.5: A transducer that increments a binary number.

Suppose $L = \{101001\}$, 37_{10} in little endian binary. Note here that we could have chosen any positive number to see the upcoming result; 37 is just a pseudo-random choice. What is $\mathrm{cl}_H(\{101001\})$? It is possible to determine this with the use of the proof system 4.2.1 or the step-by-step method in section 4.3. Both approaches are somewhat tedious and prone to error since they require some non-obvious uses of these methods. However, it is easy to see what the closure will be with transducers. We know that by theorem 5.2.1 we need to take a look at the ω-iteration. It is not hard to determine what the ω-iteration will be. What happens if we iterate T_{bin} only twice? The first iteration increments 101001 to 011001. The second iteration increments 011001 to 111001. Thus T_{bin}^2 adds 2 to a binary number. If we apply T_{bin} another time, we increment 111001 and get 000101. We have used T_{bin} three times and incremented 101001 three times as well. It may not seem surprising that the ω-iteration yields the language of all little-endian binary numbers greater than or equal to 37_{10}. Hence the closure $\mathrm{cl}_H(\{101001\})$ contains all little endian binary numbers greater than or equal to 37_{10}.

The demonstration shows that transducers can be more efficient due to their well-understood and straightforward behavior. We may even think of more examples where we have a transducer performing some translation, and the closure is intuitively found by reasoning with the ω-iteration.

Chapter 6

Related work

Other research can be classified into either two groups. On the one hand, properties of iterated transducers or specific classes of transducers are considered. The research discusses, for instance, different representations of a transducer[8]. Other examples include an algorithm that tries to reduce the size of an infinite-state ω-iterated transducers.

On the other hand, H-closures are used in Kleene algebra with hypotheses completeness theorems. In fact, completeness of Kleene algebra with hypotheses is defined as "$\mathrm{cl}_H([\![e]\!]) = \mathrm{cl}_H([\![f]\!])$ implies that $e \leq f$ is derivable from the Kleene algebra axioms and the hypotheses in H" [9]. Several instances of such completeness theorems rely on lemma's about the closures described in [5].

Both these directions discuss only transducers or H-closures. We researched the connections between the iterated transducers and the H-closures. As far as we know, we are the first to do so. Below we discuss the details of these different directions.

Length-preserving transducers

Latteux et al. [8] discuss the length-preserving transducers, the smallest class of transducers. Here Latteux shows the equivalence between several families of transductions. For instance, they showed that the smallest family of length-preserving transductions, closed under union, composition and iteration ($\mathrm{Rat}(\mathcal{T})$) coincides with the family $\mathcal{T}\mathcal{T}^{<\omega}\mathcal{T}$, the family of transductions of the form $f_{T_1} \circ f_{T_2^{<\omega}} \circ f_{T_3}$ where T_1, T_2 and T_3 are length-preserving transducers. Their main result comes down to the representation theorem, where the equivalences with the statement "T is in $\mathrm{Rat}(\mathcal{T})$" are shown. This thesis shows a similar result regarding the equivalence between functions. Although we do not specifically consider length-preserving transducers in this thesis, we show that transducers and H-closures are alike in specific circumstances.

Size reduction of $T^{<\omega}$

In the general case, $T^{<\omega}$ has an infinite amount of states. Dennis Dams et al. developed an algorithm in [4] to reduce the size to possibly a finite-state transducer. They iteratively approximate $T^{\leq n}$ and at each approximation, they compute the needed bisimulations. Eventually, the result of this iteration leads to a quotient of $T^{<\omega}$ that is semantically equivalent to $T^{<\omega}$ but reduced in size. Their result shows a different representation for some ω-iterated transducers. Our result is similar yet different because we also showed a different representation for some ω-iterated transducers. However, the difference is that we compare ω-iterated transducer with closures under hypotheses. This research, therefore, bridges a connection between transducers and language closures, while [4] deepens the knowledge about the transducer.

H-closure properties

Although [5] does not primarily focus on H-closures themselves, Doumane et al. describe H-closures and their properties are proven. With these properties, Doumane et al. prove different results regarding the decidability and complexity of Kleene algebras with hypotheses. H-closures are a vital part of that contribution. Our contribution provides a new property of the H-closures. We do not discuss Kleene algebras at all, but we provide some new insight regarding Kleene algebras with hypotheses.

Chapter 7

Conclusion

Transducers and H-closures look pretty different at first glance. Transducers are a form of generalized automata with output, and they induce a function from words to languages. H-closures are language closures under a set of hypotheses. We took both notions under the loop and showed that they are the same under specific circumstances. The research led to two theorems about how transducers and H-closures are related. The first of the two theorems regards the construction of hypotheses. If we are given a transducer T, then we can modify T to a transducer T_c such that it may leave letters untouched. With the modified transducer, we can determine the closure of a language L with $H = \{T_c(w) \leq w \mid w \in \Sigma^*\}$ by applying $f_{T_c^{<\omega}}$ to L.

This theorem is demonstrated in section 5.4.2. It shows we can always create a set of hypotheses H such that the ω-iteration of a transducer and the H-closure coincide. Therefore it allows for intuitive reasoning about what the closure might be. However, such a ω-iteration of a transducer may be an infinite state machine.

The second theorem assumes we are given a set of hypotheses H with inequations of the form $e \leq w$, where e_i is a regular expression. With this set, we can create a transducer such that applying $f_{T^{<\omega}}$ to a language L, we get the closure of L with H. We can use the second theorem in situations where we want to derive the H-closure given a set of hypotheses of the form $H = \{e_i \leq w \mid w \in \Sigma^* \wedge i \in \mathbb{N}\}$. However, using the closure itself is counterintuitive. Then it is helpful to know that we can construct a transducer that can also calculate the H-closure.

Both theorems may provide new insights or methods to derive closures, or we can use the theorems for something completely different. It is needless to say that the theorems enlighten a deeper relation between the finite-state transducers and the H-closures.

A possible continuation of the work done is to look at the iteration of transducers beyond the ω-iteration. The step-by-step method described in 4.3 could take more than a countable number of steps. After the limit step, we can take another step. We can take steps for every ordinal number, and by performing all those steps, we can determine the H-closure. Is it possible to do the same with the iteration of transducers? Can we iterate past ω iterations of a transducer T? What will this mean for the relationship between the H-closures and transducers if we can iterate T more than ω times? These questions could be investigated in future research on this subject.

Bibliography

[1] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling. 1: Parsing.* Prentice-Hall, 1972.

[2] James A. Anderson. *Automata theory with modern applications.* Cambridge University Press, 2006.

[3] Ernie Cohen. Hypotheses in kleene algebra. Technical report, 1994.

[4] Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. *J. Log. Algebraic Methods Program.*, 52-53:109–127, 2002.

[5] Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. Kleene algebra with hypotheses. In *FoSSaCS*, volume 11425 of *Lecture Notes in Computer Science*, pages 207–223. Springer, 2019.

[6] Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965.

[7] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[8] Michel Latteux, David Simplot, and Alain Terlutte. Iterated length-preserving rational transductions. In *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 1998.

[9] Damien Pous, Jurriaan Rot, and Jana Wagemaker. On tools for completeness of kleene algebra with hypotheses. In *RAMiCS*, volume 13027 of *Lecture Notes in Computer Science*, pages 378–395. Springer, 2021.

[10] Jeffrey O. Shallit. *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press, 2008.