Kleene Algebra — Lecture 4

Tobias Kappé

January 2022

1 Today's lecture

In the previous lecture, we saw that we can use a model as a (sufficient but perhaps not necessary) test for inequivalence. For instance, if $[\![e]\!]_\sigma \neq [\![f]\!]_\sigma$ for some interpretation σ, then certainly $e \not\equiv f$. The only problem was that you still had to come up with a suitable interpretation. The language model, where $[\![e]\!]_{\mathbb{E}} \neq [\![f]\!]_{\mathbb{E}}$ implies $e \not\equiv f$, does not force you to make such a choice.

However, now we have ourselves a different problem: how do we find out whether two expressions have the same language? Unlike a relation, which is always finite for a finite domain, a language may be infinite as soon as there is a Kleene star involved in the expression. Clearly, we need something more sophisticated than just calculating both sets, and comparing their contents.

In this lecture, we will see one particularly effective way to compare languages generated by expressions: converting both expressions into abstract machines called *finite automata*, and then comparing those automata.

2 Finite Automata

Let's spend some time talking about automata. Informally, an automaton is a "machine" that can be in one of several *states*. At each of those states, the automaton can read a letter from the input buffer, which then uniquely determines its *next state*. When there is no input left, the input is *accepted* if the state is *accepting*, and *rejected* otherwise. The set of words accepted when starting from a designated *initial state*, is the *language* of that state.

We can formalize these ideas precisely, as follows.

Definition 4.1. An *automaton* is a tuple $A = \langle Q, F, \delta \rangle$, where

- Q is a set of *states*, with $F \subseteq Q$ the *accepting states*; and
- $\delta : Q \times \Sigma \to Q$ is the *transition function*; and
- $q^0 \in Q$ is the *initial state*.

Define $\delta^* : Q \times \Sigma^* \to Q$ as follows:

$$\delta^*(q, \epsilon) = q \qquad \delta^*(q, \mathtt{a}w) = \delta^*(\delta(q, \mathtt{a}), w)$$

Now, the *language* of q in A, denoted $L_A(q)$, is given by $\{w : \delta^*(q, w) \in F\}$. When the automaton is clear from context, we will drop the subscript A.

Finally, we say A is finite when it has finitely many states.

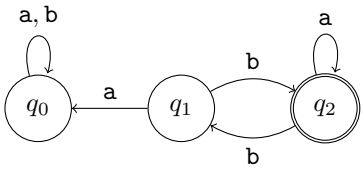


Figure 1: A drawing of an automaton.

Usually, it is easier to draw an automaton than to define it in terms of its constituent parts. In that case, we draw a circle for every state (labelled with some name), and edges between states to signify the transition function. Accepting states have double circles. For example, the automaton drawn in Figure 1 as three states: q_0, q_1, and q_2, with q_2 the sole accepting state. Its transition function can be read from the edges; for instance, $\delta(q_1, \mathtt{a}) = q_0$.

You can get an idea of $L(q_1)$ by looking at the picture: it's the set of all words that can be "read" along the (possibly looping) paths from q_1 to q_2. For instance, $\mathtt{b}, \mathtt{baabb} \in L(q_1)$. In general, it looks like the language of q_1 contains words where $\mathtt{a}$ can occur only between strictly odd and even appearances of $\mathtt{b}$.

So, how do we decide whether two states in two automata (or the same automaton) accept the same language? That is where bisimulations come in.

Definition 4.2. Let $A_i = \langle Q_i, F_i, \delta_i \rangle$ be an automaton for $i \in \{1, 2\}$. A *bisimulation* on A is a relation $R \subseteq Q_1 \times Q_2$, such that for all $q_1 \; R \; q_2$, we have:

1. $q_1 \in F_1$ if and only if $q_2 \in F_2$, and
2. for all $\mathtt{a} \in \Sigma$, it holds that $\delta_1(q_1, \mathtt{a}) \; R \; \delta_2(q_2, \mathtt{a})$.

Two states are *bisimilar* if there exists a bisimulation that relates them.

You may have encountered the idea of a bisimulation in different courses as well. The idea is that bisimilar states are "indistinguishable" in terms of their dynamics. The same can be said for the definition above, where bisimilar states are equally accepting, and this property is invariant with respect to taking transitions. As it turns out, bisimilarity and language equivalence coincide.

Lemma 4.3. *Let $A_i = \langle Q_i, F_i, \delta_i \rangle$ be an automaton for $i \in \{1, 2\}$, with $q_1 \in Q_1$ and $q_2 \in Q_2$. Now $L(q_1) = L(q_2)$ if and only if q_1 is bisimilar to q_2.*

Proof. For the forward implication, we choose R such that $q_1' \; R \; q_2'$ if and only if $L(q_1') = L(q_2')$. Clearly, $q_1 \; R \; q_2$. We now claim that R is a bisimulation. For the first condition, note that $q_1' \in F$ if and only if $\epsilon \in L(q_1') = L(q_2')$, which holds precisely when $q_2' \in F$. For the second condition, we should show that if $L(q_1') = L(q_2')$, then $L(\delta_1(q_1', \mathtt{a})) = L(\delta_2(q_2', \mathtt{a}))$ for all $\mathtt{a} \in \Sigma$. Now, if $w \in L(\delta_1(q_1', \mathtt{a}))$, then $\mathtt{a}w \in L(q_1') = L(q_2')$, and hence $w \in L(\delta_2(q_2', \mathtt{a}))$. The converse inclusion follows by a similar argument.

For the other implication, let R be a bisimulation with $q_1 \; R \; q_2$. We show, by induction on $w \in \Sigma^*$, that $w \in L(q_1')$ if and only if $w \in L(q_2')$, for all $q_1' \; R \; q_2'$. In the base, $w = \epsilon$, and hence $q_1' \in F$ if and only if $q_2' \in F$, meaning that $\epsilon \in L(q_1')$ if and only if $\epsilon \in L(q_2')$. For the inductive step, let $w = \mathtt{a}w'$ for some $\mathtt{a} \in \Sigma$. Now $w \in L(q_1')$ if and only if $w' \in L(\delta(q_1', \mathtt{a}))$. By induction (since $|w'| < |w|$), the latter is equivalent to $w' \in L(\delta(q_2', \mathtt{a}))$, and hence to $w \in L(q_2')$. □

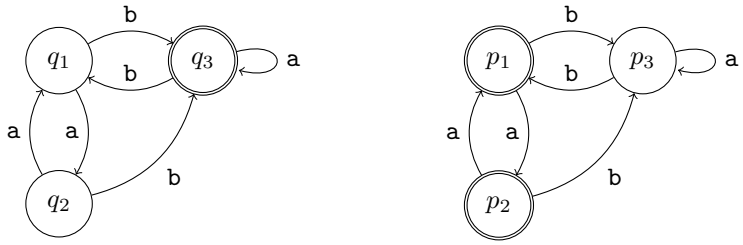


Figure 2: Two more automata.

This tells us that bisimilarity and language equivalence coincide — if we can decide one, we can decide the other. But how do we decide bisimilarity? The answer is simpler than you might expect: *try to construct the smallest bisimulation containing those states*. If this process succeeds, then we have found a bisimulation and hence a proof of language equivalence; otherwise, a bisimulation cannot exist, and thus we conclude that the languages differ.

Let's see how this works on an example[1] first, before giving the general algorithm. Consider the automata in Figure 2. Suppose we wanted to show that q_1 is language equivalent to p_3; to this end, we need to demonstrate a bisimulation R such that $q_1 \; R \; p_3$. Let's (naively) try $R = \{\langle q_1, p_3 \rangle\}$. This relation satisfies the first requirement, but not the second one, which says that $\delta(q_1, \mathtt{a}) \; R \; \delta(p_3, \mathtt{a})$ and $\delta(q_1, \mathtt{b}) \; R \; \delta(p_3, \mathtt{b})$ should hold. However, we can remedy this by adapting our guess to $R = \{\langle q_1, p_3 \rangle, \langle q_2, p_3 \rangle, \langle q_3, p_1 \rangle\}$. This candidate bisimulation still satisfies the first requirement, but is not quite a bisimulation, since $\delta(q_3, \mathtt{a}) \not R \; \delta(p_1, \mathtt{a})$. We can remedy this by adding $\langle q_3, p_2 \rangle$, obtaining

$$R = \{\langle q_1, p_3 \rangle, \langle q_2, p_3 \rangle, \langle q_3, p_1 \rangle, \langle q_3, p_2 \rangle\}$$

This final choice of R is a bisimulation, thus witnessing that $L(q_1) = L(p_3)$.

For a negative example, suppose you wanted to check whether q_1 is language equivalent to p_1. We would then need to construct a bisimulation R such that $q_1 \; R \; p_1$. Clearly, if such a bisimulation exists, it would violate the first requirement, since p_1 is accepting while q_1 is not. It follows that such a bisimulation cannot exist, and hence q_1 does not accept the same language as p_1.

To fully automate the search for a bisimulation, you can use Algorithm 1. This procedure maintains two sets: R contains the pairs that are part of our bisimulation, while T contains a "todo list" of pairs that are yet to be added. In each iteration, the algorithm removes a pair from T, checks if it is not already in R, and if the elements of the pair agree on acceptance. If both checks succeed, the pair is added to R, and its successors are added to T, to be checked later. Otherwise, if the elements of the pair do not agree on acceptance, we have reached a contradiction: the bisimulation we are looking for cannot exist; the algorithm then returns **false**. Finally, when T is empty, there are no more pairs to consider. In that case, R is a bisimulation, and the algorithm returns **true**.

Algorithm 1 is correct, for *finite* automata. The proof is not the focus of this lecture; instead, it will be part of today's homework as an optional exercise.

[1] I pulled this example from Jurriaan Rot's lecture notes on coalgebra, available online here: `http://cs.ru.nl/~jrot/coalg18/coalg-lect6.pdf`

```
Data: automata ⟨Q_i, F_i, δ_i⟩ with state q_i ∈ Q_i, for i ∈ {1, 2}.
Result: true if q_1 is bisimilar to q_2, false otherwise.
R ← ∅; T ← {⟨q_1, q_2⟩};
while T ≠ ∅ do
    pop ⟨q'_1, q'_2⟩ from T;
    if ⟨q'_1, q'_2⟩ ∉ R then
        if q'_1 ∈ F_1 ⟺ q'_2 ∈ F_2 then
            add ⟨q'_1, q'_2⟩ to R;
            add ⟨δ_1(q'_1, a), δ_2(q'_2, a)⟩ to T for all a ∈ Σ;
        else
            return false;
return true;
```

Algorithm 1: Bisimulation search.

3 Brzozowski's construction

By now, we know how to compare two states of an automaton for language equivalence. However, we wanted to figure out if two *expressions* denote the same language. This means that, if we can convert an expression to an automaton with a state that accepts the same language, we have reached our goal. There is a wide variety of such constructions. Today, we will go over one that works well for the material covered later this week: *Brzozowski's construction.*

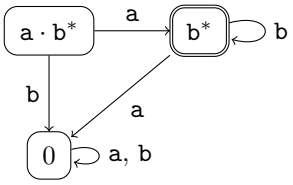


Figure 3: An automaton for the expression $\mathtt{a} \cdot \mathtt{b}^*$.

The idea behind Brzozowski's construction is to create an infinite automaton, where each state is an expression representing the language to be accepted. For example, consider $\mathtt{a} \cdot \mathtt{b}^*$. Brzozowski's construction will yield an automaton very similar to the one in Figure 3. A state recognizing this language cannot be accepting, because $\epsilon \notin [\![\mathtt{a} \cdot \mathtt{b}^*]\!]_{\mathbb{E}}$. Also, since no word in $[\![\mathtt{a} \cdot \mathtt{b}^*]\!]_{\mathbb{E}}$ starts with a $\mathtt{b}$, its $\mathtt{b}$-transition must go to a state that does not accept any word — such as the one represented by the expression 0. On the other hand, upon reading the letter $\mathtt{a}$, the remaining words to be read before we can accept must come from the language denoted by $\mathtt{b}^*$, and so we transition to the state for that expression. Since $\epsilon \in [\![\mathtt{b}^*]\!]_{\mathbb{E}}$, this state is accepting. However, since no word in $[\![\mathtt{b}^*]\!]_{\mathbb{E}}$ begins with a $\mathtt{a}$, the $\mathtt{a}$-transition again leads to 0. On the other hand, after reading a $\mathtt{b}$, the remaining words also come from $\mathtt{b}$, and so this state loops to itself.

Let's figure out which expressions need to give rise to accepting states.

Definition 4.4. We define $\mathbb{A}$ as the smallest subset of $\mathbb{E}$ satisfying the rules

$$\frac{}{1 \in \mathbb{A}} \qquad \frac{e \in \mathbb{A} \qquad f \in \mathbb{E}}{e + f, f + e \in \mathbb{A}} \qquad \frac{e, f \in \mathbb{A}}{e \cdot f \in \mathbb{A}} \qquad \frac{e \in \mathbb{E}}{e^* \in \mathbb{A}}$$

It is not too hard to show that $\mathbb{A}$ characterizes expressions whose language contains the empty set, i.e., that for $e \in \mathbb{E}$ we have $e \in \mathbb{A}$ if and only if $\epsilon \in [\![e]\!]_{\mathbb{E}}$. We omit this proof, and instead prove something slightly stronger in a moment. Before we can do that, we must define the transition structure of our automaton.

Definition 4.5. We define $d : \mathbb{E} \times \Sigma \to \mathbb{E}$ inductively, as follows:

$$d(0, \mathtt{a}) = 0 \qquad d(1, \mathtt{a}) = 0 \qquad d(\mathtt{b}, \mathtt{a}) = [\mathtt{b} = \mathtt{a}] \qquad d(e + f, \mathtt{a}) = d(e, \mathtt{a}) + d(f, \mathtt{a})$$

$$d(e \cdot f, \mathtt{a}) = d(e, \mathtt{a}) \cdot f + [e \in \mathbb{A}] \cdot d(f, \mathtt{a}) \qquad d(e^*, \mathtt{a}) = d(e, \mathtt{a}) \cdot e^*$$

Here, $[\Phi]$ is shorthand for 1 when Φ holds, and 0 otherwise.

Intuitively, $d(e, \mathtt{a})$ is the expression denoting words that, prepended with $\mathtt{a}$, constitute a word in e. So, for instance, $d(\mathtt{a}, \mathtt{a}) = 1$, because after reading $\mathtt{a}$, we need only read a word from 1 (i.e., ϵ). Similarly, the definition of $d(e \cdot f, \mathtt{a})$, encodes that reading a word from $e \cdot f$ leaves us with two choices: either (1) we read a word from e, and proceed with what is left of that followed by f, or (2) e accepts the empty word, and so we can "skip" it, reading from f instead.

In the literature, you'll often see $d(e, \mathtt{a})$ referred to as the $\mathtt{a}$-*derivative of* e. Part of the reason for this nomenclature is an analogy with calculus, where a function can be reconstructed by integrating its derivatives, and possibly adding a constant. The same is true for expressions, in the sense that e can be reconstructed from its derivatives, as witnessed by the following result.

Theorem 4.6 (Fundamental theorem)**.** *Let $e \in \mathbb{E}$. The following holds:*

$$e \equiv [e \in \mathbb{A}] + \sum_{\mathtt{a} \in \Sigma} \mathtt{a} \cdot d(e, \mathtt{a})$$

We use the generalized sum notation in the same way that you would expect. Here (and in other contexts), the order and bracketing of the terms does not matter: since $+$ is commutative and associative with respect to $\equiv$, whether the equivalence holds is independent of our choices in that department. The proof of Theorem 4.6 proceeds by induction on e, and is part of today's homework.

Lemma 4.7. *Let $B = \langle \mathbb{E}, \mathbb{A}, d \rangle$. Now $L_B(e) = [\![e]\!]_{\mathbb{E}}$.*

Proof. Let $w \in \Sigma^*$. We prove that $w \in L(e)$ if and only if $w \in [\![e]\!]_{\mathbb{E}}$ for all $e \in \mathbb{E}$ by induction on w. In the base, $w = \epsilon$. By Theorem 4.6 and soundness, we find that $\epsilon \in [\![e]\!]_{\mathbb{E}}$ if and only if $e \in \mathbb{A}$; the latter is equivalent to $\epsilon \in L(e)$.

For the inductive step, let $w = \mathtt{a}w'$. By Theorem 4.6 and soundness, $w \in [\![e]\!]_{\mathbb{E}}$ if and only if $w' \in [\![d(e, \mathtt{a})]\!]_{\mathbb{E}}$. By induction, the latter is equivalent to $w' \in L(d(e, \mathtt{a}))$, and hence to $w = \mathtt{a}w' \in L(e)$. □

There is just one last thing to sort out. In the first part of the lecture, we noted that the bisimilarity checking algorithm words for *finite* automata. But the automaton A as studied in the lemma above is not finite — it contains all (infinitely many) expressions $\mathbb{E}$! As a work-around, you could try and restrict the automaton to the states reachable from e. Unfortunately for us, this does not help; for instance, if $e = \mathtt{a}^*$, then states reachable from e include

$$1 \cdot \mathtt{a}^* \qquad 0 \cdot \mathtt{a}^* + 1 \cdot \mathtt{a}^* \qquad 0 \cdot \mathtt{a}^* + 0 \cdot \mathtt{a}^* + 1 \cdot \mathtt{a}^* \qquad \ldots$$

Clearly, there are infinitely many of these.

As you can tell, part of the problem in the bad case above lies in the duplication of terms. Clearly, $0 \cdot \mathtt{a}^* + 1 \cdot \mathtt{a}^*$ denotes the same language as $0 \cdot \mathtt{a}^* + 0 \cdot \mathtt{a}^* + 1 \cdot \mathtt{a}^*$. If we could somehow eliminate duplicate terms when they pop up, maybe we could obtain a finite automaton. The next notion tries to do just that.

Definition 4.8. We define $\sim$ as the smallest equivalence on $\mathbb{E}$ satisfying

$$e \sim e + e \qquad e + f \sim f + e \qquad e + (f + g) \sim (e + f) + g$$

for all $e, f, g \in \mathbb{E}$. We write $[e]$ for the $\sim$-equivalence class of e, i.e., $\{e' \in \mathbb{E} : e \sim e'\}$. Also, we write $\hat{\mathbb{E}}$ for the quotient of $\mathbb{E}$ by $\sim$, i.e., $\{[e] : e \in \mathbb{E}\}$.

We should note that $\sim$ is an equivalence relation, but not a congruence. So, for instance, $(e + f) \cdot g \sim (f + e) \cdot g$ does *not* hold in general. We could define $\sim$ to be a congruence, if we wanted to, but that would just complicate our proofs.

It turns out that $\mathbb{A}$ and d are invariant w.r.t. $\sim$, in the following sense.

Lemma 4.9. *Let $e, f \in \mathbb{E}$ with $e \sim f$. The following hold:*

(i) $e \in \mathbb{A}$ if and only if $f \in \mathbb{A}$

(ii) for all $\mathtt{a} \in \Sigma$, we have $d(e, \mathtt{a}) \sim d(f, \mathtt{a})$.

Proof sketch. We need only check the three rules generating $\sim$. For instance, if $e + f \in \mathbb{A}$ because $e \in \mathbb{A}$ or $f \in \mathbb{A}$, then $f + e \in \mathbb{A}$ as well. For the second claim, we note that $d(e + f, \mathtt{a}) = d(e, \mathtt{a}) + d(f, \mathtt{a}) \sim d(f, \mathtt{a}) + d(e, \mathtt{a}) = d(f + e, \mathtt{a})$. □

We write $\hat{\mathbb{A}}$ for $\{[e] : e \in \mathbb{A}\}$ define $\hat{d} : \hat{\mathbb{E}} \times \Sigma \to \hat{\mathbb{E}}$ by setting $\hat{d}([e], \mathtt{a}) = [d(e, \mathtt{a})]$. By the lemma above, these definitions are unambiguous. This yields a new automaton, given by $\hat{B} = \langle \hat{\mathbb{E}}, \hat{\mathbb{A}}, \hat{d} \rangle$, accepting the same languages.

Lemma 4.10. *Let $e \in \mathbb{E}$; now $L_{\hat{B}}([e]) = L_B(e)$.*

Proof. Let $R = \{\langle e, [e] \rangle : e \in \mathbb{E}\}$. We claim that R is a bisimulation between $\langle \mathbb{E}, \mathbb{A}, d \rangle$ and $\langle \hat{\mathbb{E}}, \hat{\mathbb{A}}, \hat{d} \rangle$. To see this, note that $e \in \mathbb{A}$ if and only if $[e] \in \hat{\mathbb{A}}$, by definition. Furthermore, for all $\langle e, [e] \rangle \in R$, we have that $\langle d(e, \mathtt{a}), [d(e, \mathtt{a})] \rangle \in R$ for all $\mathtt{a} \in \Sigma$, also by definition. The claim then follows by Lemma 4.3. □

So, our quotiented automaton also has a state that accepts $[\![e]\!]_{\mathbb{E}}$ for every $e \in \mathbb{E}$, namely $[e]$. But is it also the case that every state can reach only finitely many states? The following (somewhat technical) lemma settles this question.

Lemma 4.11. *Let $e \in \mathbb{E}$. The set $r(e) = \{\hat{d}^*([e], w) : w \in \Sigma^*\}$ is finite.*

Proof sketch. By induction on e. We highlight the inductive case $e = e_1 \cdot e_2$, where the claim holds for e_1 and e_2. By induction on $w \in \Sigma^*$, we can show that $d^*(e, w) \sim f_1 + f_2 + \cdots + f_n$, where for $1 \leq i \leq n$ either $f_i = d^*(e_1, w) \cdot e_2$, or $f_i = b \cdot d^*(e_2, w)$ where $b \in \{0, 1\}$. By induction, there are finitely many choices for f_i, let's say N. Furthermore, without loss of generality, $n \leq N$; after all, if $n > N$, then we have some duplicate terms f_i and f_j, which can be eliminated by $\sim$. It follows that there only finitely many possibilities for $d^*(e, w)$ up to $\sim$, and hence only finitely many $\hat{d}^*([e], w) = [d^*(e, w)]$. □

Thus, if we want to create an automaton accepting $[\![e]\!]_{\mathbb{E}}$ for some $e \in \mathbb{E}$, we can simply take the restriction of $\langle \hat{\mathbb{E}}, \hat{\mathbb{A}}, \hat{d} \rangle$ to $r(e)$. Since this set is closed, $\hat{d}$ is well-defined — i.e., if $[f] \in r(e)$ and $\mathtt{a} \in \Sigma$, then $\hat{d}([f], \mathtt{a}) \in r(e)$. Moreover, because excluding states unreachable from $[e]$ does not change its language, the finite automaton also satisfies our objective.

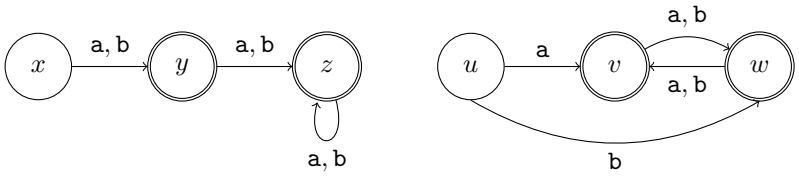
4 Homework

1. Create an automaton for each of the following languages over $\Sigma = \{\mathtt{a}, \mathtt{b}\}$:

   (a) The language L_0 of words where every third letter is an $\mathtt{a}$, unless it is preceded by a $\mathtt{b}$. Examples of words in L_0 include $\mathtt{aaa}$, ϵ, $\mathtt{b}$, and $\mathtt{bbb}$. Examples of words *not* in L_0 include $\mathtt{aab}$ and $\mathtt{aaabab}$.

   (b) *(Optional)* The language L_1 of words that do not contain three subsequent $\mathtt{a}$'s or $\mathtt{b}$'s. Examples of words in L_1 include ϵ, $\mathtt{ab}$ and $\mathtt{abaa}$. Examples of words *not* in L_1 include $\mathtt{aaa}$ and $\mathtt{abbba}$.

2. Consider the automata drawn below:



   Prove that x and u are bisimilar.

3. *(Optional)* Prove that the following three properties are *invariants* of the **while-do** loop in Algorithm 1 — that is to say: they are true *before* the loop starts, and they are preserved every time the loop body runs in full:[2]

   (a) If $\langle q_1', q_2' \rangle \in R$, then $q_1' \in F_1$ if and only if $q_2' \in F_2$.

   (b) If $\langle q_1', q_2' \rangle \in R$, then $\langle \delta_1(q_1', \mathtt{a}), \delta(q_2', \mathtt{a}) \rangle \in R \cup T$ for all $\mathtt{a} \in \Sigma$.

   (c) If R' is a bisimulation relating q_1 to q_2, then $R \cup T \subseteq R'$.

   Show that, when the loop terminates, these properties together with $T = \emptyset$ imply that R is the *smallest* bisimulation relating q_1 to q_2.

4. *(Optional)* Show that Algorithm 1 terminates if Q_1 and Q_2 are finite.

5. Prove the fundamental theorem (Theorem 4.6) by induction on e. Your induction hypothesis should be that the claim holds for all direct subexpressions of e. For instance, for the case $e = e_0 \cdot e_1$, you may assume

$$e_i \equiv [e_i \in \mathbb{A}] + \sum_{\mathtt{a} \in \Sigma} \mathtt{a} \cdot d(e_i, \mathtt{a})$$

   Hint: for the case where $e = e_0^*$, it is useful to first prove that, for all $f \in \mathbb{E}$ and $b \in \{0, 1\}$, it holds that $(b + f)^* \equiv f^*$.

6. Use Brzozowski's construction to come up with an automaton for the expression $\mathtt{b}^* \cdot \mathtt{a}$. Show your work by first computing the values of $\hat{d}([\mathtt{b}^* \cdot \mathtt{a}], \mathtt{c})$ for each $\mathtt{c} \in \Sigma$. Then, repeat this process for all new states/expressions that arise, until there are no new states.

7. *(Optional)* Construct the Brzozowski automaton for $e = \mathtt{a} + \mathtt{a} \cdot \mathtt{b} + \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b}^*$, and show that the state for e is bisimilar to the state $\mathtt{a} \cdot \mathtt{b}^*$ in Figure 3.

[2] In particular, this means that the **return** statement is not executed.

5 Bibliographical notes

Automata developed over time, but some of the very first ideas appeared in the work of McCulloch and Pitts [MP43]. Moore [Moo56] observed that language equivalence of finite automata is decidable. Another algorithm to achieve the same is due to Hopcroft [Hop71]. The algorithm presented here is a simplified version of a more sophisticated algorithm due to Hopcroft and Karp [HK71].

Kleene [Kle56] first observed the connection between rational expressions and automata, although his presentation of the automata was rather different (and well worth a read). Brzozowski's construction [Brz64], presented in this lecture, is a particularly elegant method. Later constructions include an inductive construction by Thompson [Tho68] and a refinement of Brzozowski's method by Antimirov [Ant96]. For a good overview of the different constructions, we refer to Watson [Wat93].

References

[Ant96] Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. doi:10.1016/0304-3975(95)00182-4.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

[HK71] John E. Hopcroft and Richard M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report TR 71-114, Cornell University, Ithaca, NY, December 1971.

[Hop71] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971. doi:10.1016/B978-0-12-417750-5.50022-1.

[Kle56] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–154, 1956. doi:10.1515/9781400882618-006.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5(4):115–133, 1943. doi:10.1007/BF02478259.

[Tho68] Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.

[Wat93] Bruce W. Watson. A taxonomy of finite automata construction algorithms. Technical report, Technische Universiteit Eindhoven, 1993. URL: https://research.tue.nl/files/2482472/9313452.