Kleene Algebra — Lecture 3

Tobias Kappé

ESSLLI 2023

1 Today's lecture

If you want to check whether two rational expressions have the same semantics, you can try proving them equivalent using the laws of Kleene algebra. As we have seen, however, you may get stuck for one of two reasons: either you are trying to prove an equivalence that does not follow from the axioms (but may still be true), or the equivalence you are trying to prove is false. In this lecture, we will see a particular method to algorithmically check, given rational expressions $e, f \in \mathbb{E}$, whether $[\![e]\!]_{\mathbb{E}} = [\![f]\!]_{\mathbb{E}}$. Importantly, the algorithm that we will develop will be able to give you a definitive answer to this question; this means that you will be able to eliminate the second possibility.

The methodology that we will employ is commonplace throughout computer science, and so it is good to lay out explicitly. First, we will introduce *automata* as a finite way to represent a language. Next, we will study a structural property of automata that characterizes language equivalence. Finally, we will develop a construction that turns an expression e into an automaton that represents $[\![e]\!]_{\mathbb{E}}$. In total, this will give us a recipe for checking whether $[\![e]\!]_{\mathbb{E}} = [\![f]\!]_{\mathbb{E}}$: simply convert both e and f into automata, and check those for language equivalence.

2 Finite 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 more precisely, as follows.

Definition 3.1. An *automaton* is a tuple $A = \langle Q, \to, I, F \rangle$, where

- Q is a set of *states*; and
- $\to \subseteq Q \times \Sigma \times Q$ is the *transition relation*; and
- $I, F \subseteq Q$ are the *initial* and *accepting states* respectively

We use $q \xrightarrow{\mathtt{a}} q'$ to denote that $\langle q, \mathtt{a}, q' \rangle \in \to$. Now, the *language* of q in A, denoted $L_A(q)$, is the smallest set satisfying the following rules

$$\frac{q \in F}{\epsilon \in L_A(q)} \qquad \frac{w \in L_A(q') \qquad q \xrightarrow{\mathtt{a}} q'}{\mathtt{a}w \in L_A(q)}$$

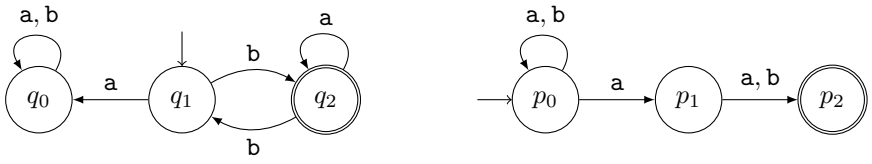


Figure 1: Graphical depictions of a deterministic automaton (on the left), and a non-deterministic automaton (on the right).

When the automaton is clear from context, we will drop the subscript. The *language of* A, denoted $L(A)$, is the union of $L_A(q)$ for all $q \in I$.

An automaton is *finite* when it has finitely many states, and *deterministic* when (1) I is a singleton set, and (2) for each $q \in Q$ and $\mathtt{a} \in \Sigma$, there is *exactly* one $q' \in Q$ such that $q \xrightarrow{\mathtt{a}} q'$. For deterministic automata, we may also write q_I for the sole initial state, and $(q)_\mathtt{a}$ for the unique state such that $q \xrightarrow{\mathtt{a}} (q)_\mathtt{a}$.

Sometimes is easier to draw an automaton to define it. In that case, we draw a circle or rrectanglefor every state (labelled with some name), and edges between states to signify the transition function. Accepting states have their contours doubled. For example, the automaton drawn in Figure 1 on the left has three states: q_0, q_1, and q_2, with q_2 the sole accepting state. Its transition relation can be read from the edges; for instance, $q_1 \xrightarrow{\mathtt{a}} q_2$. Initial states are indicated by an arrow without a source node, as is done for q_1 in the example. This automaton is also deterministic: there is exactly one initial state, and for every state and letter `a`, there is precisely one `a`-successor state reached through the transition relation. In contrast, the automaton drawn in Figure 1 on the right is non-deterministic: there are *two* states reachable from p_0 by `a` (p_0 itself and p_1); also, there are no states reachable from p_2 by reading `a` or `b`.

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 `a` can occur only between strictly odd and even appearances of `b`.

3 Bisimulation

So, when do two states in two automata (or even the same automaton) accept the same language? This is not immediately obvious; there can be many automata that accept the same language, and their structures can vary a lot: consider for instance the automata in Figure 2, which accept the same language but are quite different structurally. That is where (bi)simulations come in.

Definition 3.2. Let $A_i = \langle Q_i, \to_i, I_i, F_i \rangle$ be an automaton for $i \in \{1, 2\}$. A *simulation* of A_1 by A_2 is a relation $R \subseteq Q_1 \times Q_2$, s.t. for all $q_1\ R\ q_2$:

1. if $q_1 \in F_1$, then $q_2 \in F_2$; and
2. if $q_1 \xrightarrow{\mathtt{a}}_1 q_1'$, then there exists a $q_2' \in Q_2$ such that $q_2 \xrightarrow{\mathtt{a}}_2 q_2'$ and $q_1'\ R\ q_2'$.

Moreover, a *bisimulation* between A_1 and A_2 is a simulation of A_1 by A_2 such that its converse (i.e., the relation $R^c = \{\langle q_2, q_1 \rangle \in Q_2 \times Q_1 : \langle q_1, q_2 \rangle \in R\}$) is a

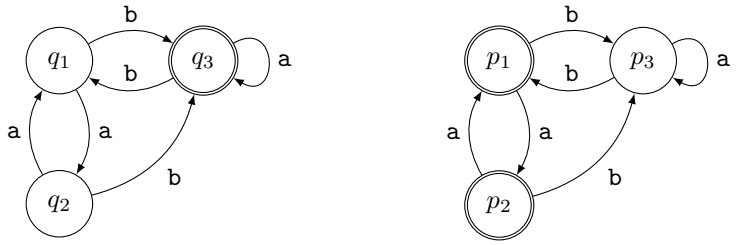


Figure 2: Two more automata.

simulation of A_2 by A_1. A state $q_1 \in Q_1$ is *(bi)similar* to a state $q_2 \in Q_2$ when there exists a (bi)simulation 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 are related.

Lemma 3.3. *Let $A_i = \langle Q_i, \to_i, I_i, F_i \rangle$ be an automaton for $i \in \{1, 2\}$, with $q_1 \in Q_1$ and $q_2 \in Q_2$. The following properties hold:*

1. *If q_1 is similar to q_2, then $L(q_1) \subseteq L(q_2)$.*
2. *If $L(q_1) \subseteq L(q_2)$, and the A_i are deterministic, then q_1 is similar to q_2.*

Proof. For the first property, 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, $w = \epsilon$. Now, if $\epsilon \in L(q_1')$, then surely $q_1' \in F$. Since $q_1' \; R \; q_2'$, it follows that $q_2' \in F$, meaning $\epsilon \in L(q_2')$. The converse follows analogously. For the inductive step, let $w = \mathtt{a}w'$ with $\mathtt{a} \in \Sigma$ such that $w' \in L(q_1'')$ and $q_1' \xrightarrow{\mathtt{a}} q_1''$. Because $q_1' \; R \; q_2'$ there exists a $q_2'' \in Q_2$ such that $q_2' \xrightarrow{\mathtt{a}} q_2''$ and $q_1'' \; R \; q_2''$. By induction, we know that $w' \in L(q_2'')$, and thus that $w = \mathtt{a}w' \in L(q_2')$.

For the forward implication, we choose $R = \{\langle q_1', q_2' \rangle \in Q_1 \times Q_2 : L_{A_1}(q_1') = L_{A_2}(q_2')\}$. Clearly, $q_1 \; R \; q_2$. We now claim that R is a bisimulation; to this end, suppose $q_1' \; R \; q_2'$. 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 $q_1' \xrightarrow{\mathtt{a}} q_1''$, then there exists a $q_2'' \in Q_2$ such that $q_2' \xrightarrow{\mathtt{a}} q_2''$ and $q_1'' \; R \; q_2''$, i.e., $L_{A_1}(q_1'') \subseteq L_{A_2}(q_2'')$. Because A_1 and A_2 are deterministic, we know that in this case $q_1'' = (q_1)_{\mathtt{a}}$, and we can choose $q_2'' = (q_2')_{\mathtt{a}}$. It remains to show that $L_{A_1}((q_1')_{\mathtt{a}}) \subseteq L_{A_2}((q_2')_{\mathtt{a}})$. To this end, let $w \in L_{A_1}((q_1')_{\mathtt{a}})$. Then we have that $\mathtt{a}w \in L_{A_1}(q_1')$, by definition, and thus $\mathtt{a}w \in L_{A_2}(q_2')$. But this can *only* be the case when $w \in L_{A_2}((q_2')_{\mathtt{a}})$, and so we are done. □

Corollary 3.4. *Let $A_i = \langle Q_i, \to_i, I_i, F_i \rangle$ be an automaton for $i \in \{1, 2\}$, with $q_1 \in Q_1$ and $q_2 \in Q_2$. The following properties hold:*

1. *If q_1 is bisimilar to q_2, then $L(q_1) = L(q_2)$.*
2. *If $L(q_1) = L(q_2)$, and the A_i are deterministic, then q_1 is bisimilar to q_2.*

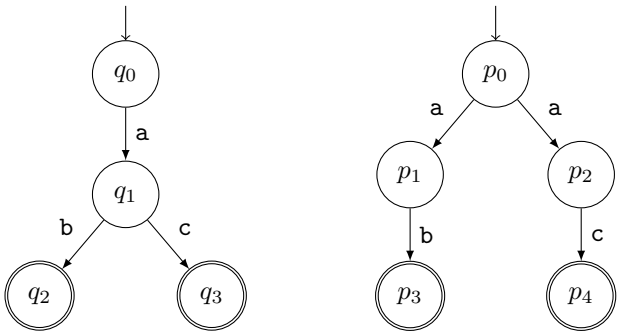


Figure 3: Language equivalence does not imply (bi)similarity in general.

This tells us that, *for deterministic automata,* bisimilarity and language equivalence coincide — if we can decide one, we can decide the other. Indeed, this dependence on determinism is necessary for the second part of Lemma 3.3 and corollary 3.4: there exist non-deterministic automata with states that have the same language, but are not bisimilar. A classic example of this is given by the automata in Figure 3: here, q_0 in the automaton on the left has the same language as p_0 in the automaton on the right, but q_0 is *not* similar to p_0, as the former can take an `a`-step to q_1, which has both a `b`-step and a `c`-step available, while p_0 cannot make a transition to a state that can offer both options.

4 Finding bisimulations

Let's see how we could try to find a bisimulation manually.[1] first, before giving the general algorithm. Consider the deterministic 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 $(q_1)_{\mathtt{a}} \; R \; (p_3)_{\mathtt{a}}$ and $(q_1)_{\mathtt{b}} \; R \; (p_3)_{\mathtt{b}}$ should hold. However, we can remedy this by updating our guess to $R = \{\langle q_1, p_3 \rangle, \langle q_2, p_3 \rangle, \langle q_3, p_1 \rangle\}$. This candidate still satisfies the first requirement, but is not quite a bisimulation, since $(q_3)_{\mathtt{a}} \not R \; (p_1)_{\mathtt{a}}$. 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

[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`

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**.

```
Data: det. automata ⟨Q_i, →_i, I_i, F_i⟩ and states 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 ⟨(q'_1)_a, (q'_2)_a⟩ to T for all a ∈ Σ;
        else
            return false;
return true;
```

Algorithm 1: Bisimulation search.

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.

5 The powerset construction

Getting back to the bigger picture for a moment, recall that we started out wanting to decide whether two expressions have the same language, and we want to do this by deciding whether two automata have the same language. This raises two questions. First, how do we even decide bisimilarity? Second, what if our automata are not deterministic? We will provide a fairly simple algorithm for the first question in the next lecture; for now, we focus on determinism.

The added value of determinism is that, given a state in an automaton and an input letter, you always only have *one choice* for the next state. We will now see that it is possible to convert a general automaton into a deterministic one accepting the same language; the key insight is that the automaton we will produce always takes *all possible successors*; as a result, the states of the new automaton will represent *sets* of the states of the old automaton.

Definition 3.5. Let $A = \langle Q, \to, I, F\rangle$ be an automaton. The *powerset automaton* of A is the *deterministic* automaton $\langle 2^Q, \to', \{I\}, F'\rangle$, where $F' = \{S \subseteq Q : S \cap F \neq \emptyset\}$ and $\to'$ is the smallest relation where for all $S \subseteq Q$, we have

$$S \xrightarrow{\mathtt{a}}{}' \{q' \in Q : \exists q \in S.\ q \xrightarrow{\mathtt{a}} q'\}$$

As an example of the powerset construction, consider the automaton on the right in Figure 1. Its powerset automaton is drawn in Figure 4; to reduce visual clutter, we have dropped the set braces from the annotation of each state. In

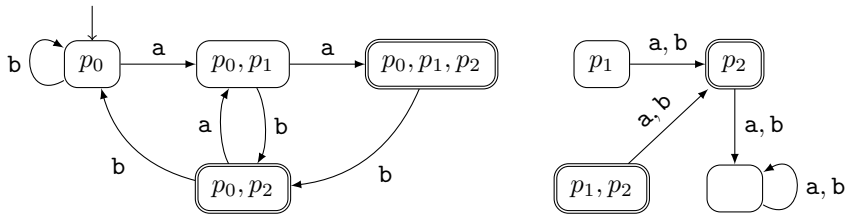


Figure 4: The powerset automaton of the automaton in Figure 1 on the right.

this automaton, instead of having two outgoing a-transitions from p_0 to p_0 and p_1, we now have a single a-transition from $\{p_0\}$ to the composite state $\{p_0, p_1\}$. This state, in turn, has a single a-transition to $\{p_0, p_1, p_2\}$, because p_0 has a a-transition to p_0 and p_1, while p_1 has a a-transition to p_2.

Remark 3.6. Although the four states on the right are not reachable from the initial state $\{p_0\}$, it is worth pointing out that every automaton obtained by the powerset construction will have a state $\emptyset$ (on the bottom right in Figure 4), which is non-accepting and transitions only to itself.

We are now in a position to prove that the powerset automaton of A accepts the very same language as A itself, by relating the language of a state in the powerset automaton to the languages of its constituent states in A, as follows.

Lemma 3.7. *Let $A = \langle Q, \to, I, F\rangle$ be an automaton, and $A' = \langle 2^Q, \to', \{I\}, F'\rangle$ its powerset automaton. For all $S \subseteq Q$ we have $L_{A'}(S) = \bigcup_{q\in S} L_A(q)$. In particular, for all $q \in Q$ we have that $L_{A'}(\{q\}) = L_A(q)$, and $L(A') = L(A)$.*

Proof. We prove that for all $w \in \Sigma^*$ that if $S \subseteq Q$, then $w \in L_{A'}(S)$ if and only if $w \in L_A(q)$ for some $q \in S$; the first claim then follows. To this end, we proceed by induction on w. In the base, where $w = \epsilon$, we have that $w \in L_{A'}(S)$ if and only if $S \in F'$, i.e., if and only if $S \cap F \neq \emptyset$, which is equivalent to the existence of some $q \in S \cap F$, or in other words $q \in S$ such that $w = \epsilon \in L_A(q)$.

For the inductive step, let $w = \mathtt{a}w'$ for some $\mathtt{a} \in \Sigma$ and $w' \in \Sigma^*$. Now $w \in L_{A'}(S)$ if and only $w' \in L_{A'}(S')$, where $S' = \{q' \in Q : \exists q \in S.\ q \xrightarrow{\mathtt{a}} q'\}$. By induction, the latter is equivalent to the existence of a $q' \in S'$ such that $w \in L_A(q')$; since $q' \in S'$, this holds precisely when there exist $q \in S$ and $q' \in Q$ with $q \xrightarrow{\mathtt{a}} q'$ and $w' \in L_A(q')$, i.e., when there exists a $q \in S$ with $w \in L_A(q)$.

The second claim follows immediately; as for the last claim, we note that

$$L(A') = \bigcup_{S\in I'} L_{A'}(S) = L_{A'}(I) = \bigcup_{q\in I} L_A(q) = L(A) \qquad \square$$

As a consequence of the above, we have a way to compare the languages of two (not necessarily deterministic) automata: simply apply the powerset construction, and verify that the resulting (deterministic) automata have bisimilar initial states. In general, the powerset construction does give us an exponential blowup in the number of states, but that is the cost of doing business.

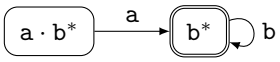


Figure 5: An automaton for the expression $\mathtt{a} \cdot \mathtt{b}^*$.

6 Antimirov's construction

By now, we know that if we want to compare two automata for language equivalence, we can try and establish a bisimulation between their powerset automata. However, we wanted to figure out if two *expressions* denote the same language. This means that we still need to figure out how to convert an expression to an automaton that accepts the same language. There is a wide variety of such constructions. Today, we will go over *Antimirov's construction*, which is itself a variation of an earlier construction due to Brzozowski.

The idea behind Antimirov's construction is to create an infinite automaton, where each state is an expression representing the language to be accepted by that state. For example, consider $\mathtt{a} \cdot \mathtt{b}^*$. A state recognizing this language cannot be accepting, because $\epsilon \notin [\![\mathtt{a} \cdot \mathtt{b}^*]\!]_{\mathbb{E}}$. 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. After reading a $\mathtt{b}$, the remaining words also come from $\mathtt{b}$, and so this state loops to itself. The resulting automaton is drawn in Figure 5; the full construction, which we will describe momentarily, will give us a very similar automaton.

First, let's look at the expressions that are attached to accepting states. Intuitively, these must be the expressions whose language includes the empty word ϵ. As it turns out, we can characterise these expressions directly, as follows.

Definition 3.8. 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.

We must also put a transition structure on our automaton. Here, the intuition is that if we are in a state labelled by e and we read an $\mathtt{a}$, then we should transition to a state labelled by an expression e' whose language includes words w such that $\mathtt{a}w$ is in the language of e. Conversely, if $\mathtt{a}w$ is a word in the language of e, then we should be able to make an $\mathtt{a}$-transition to a state labelled by an expression e' such that w is in the language of e. We do this as follows.

Definition 3.9. We define $\to_{\mathbb{E}} \subseteq \mathbb{E} \times \Sigma \times \mathbb{E}$ as the smallest relation satisfying

$$\frac{}{\mathtt{a} \xrightarrow{\mathtt{a}}_{\mathbb{E}} 1} \qquad \frac{e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'}{e + f \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'} \qquad \frac{f \xrightarrow{\mathtt{a}}_{\mathbb{E}} f'}{e + f \xrightarrow{\mathtt{a}}_{\mathbb{E}} f'}$$

$$\frac{e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'}{e \cdot f \xrightarrow{\mathtt{a}}_{\mathbb{E}} e' \cdot f} \qquad \frac{e \in \mathbb{A} \qquad f \xrightarrow{\mathtt{a}}_{\mathbb{E}} f'}{e \cdot f \xrightarrow{\mathtt{a}}_{\mathbb{E}} f'} \qquad \frac{e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'}{e^* \xrightarrow{\mathtt{a}}_{\mathbb{E}} e' \cdot e^*}$$

Example 3.10. If $e = \mathtt{a} + \mathtt{b} \cdot \mathtt{a}^*$, then $e \xrightarrow{\mathtt{a}}_{\mathbb{E}} 1$ because $\mathtt{a} \xrightarrow{\mathtt{a}}_{\mathbb{E}} 1$. Similarly, $e \xrightarrow{\mathtt{b}} 1 \cdot \mathtt{a}^*$, because $\mathtt{b} \cdot \mathtt{a}^* \xrightarrow{\mathtt{b}}_{\mathbb{E}} 1 \cdot \mathtt{a}^*$, which in turn holds because $\mathtt{b} \xrightarrow{\mathtt{b}}_{\mathbb{E}} 1$.

Intuitively, the first rule covers reading the only word in the language of $\mathtt{a}$. The second and third rule ensure that if we can read an $\mathtt{a}$ from e or f, then we can also read this $\mathtt{a}$ from $e + f$ — and hence go on to accept the words in either e or f. The fourth rule says that if we can read an $\mathtt{a}$ from e to end up in e', then we can read a $\mathtt{a}$ from $e \cdot f$ to end up in $e' \cdot f$ — intuitively, we read the beginning of a word in the language of e, so we still need to finish that word, and then go on to read a word in the language of f. The fifth rule takes care of the latter: if the language of e contains the empty word, then we can start reading a word accepted by f. The last rule says that if $\mathtt{a}$ is the start of a word accepted by e, then it is also the start of a word accepted by e^* — indeed, from e^* we move to $e' \cdot e^*$ because we need to finish reading the word accepted by e, and when we are done we can choose to read another word from e^*.

At this point, we could create for every expression e an automaton like $\langle \mathbb{E}, \to_{\mathbb{E}}, \{e\}, \mathbb{A} \rangle$. The only problem is that this automaton is not finite, as it contains a state for each expression. Fortunately, we can do a lot better if we focus on a particular finite subset of $\mathbb{E}$, defined as follows.

Definition 3.11. We define $\rho : \mathbb{E} \to 2^{\mathbb{E}}$ by induction, as follows.

$$\rho(0) = \rho(1) = \emptyset \qquad \rho(\mathtt{a}) = \{1\} \qquad \rho(e + f) = \rho(e) \cup \rho(f)$$

$$\rho(e \cdot f) = \{e' \cdot f : e' \in \rho(e)\} \cup \rho(f) \qquad \rho(e^*) = \{e' \cdot e^* : e' \in \rho(e)\}$$

Intuitively, $\rho(e)$ holds the expressions that may be reached when reading a word starting from the expression e (but not necessarily e itself). We can show that $\rho(e)$ is closed under taking transitions, in the following sense.

Lemma 3.12. *Let $e \in \mathbb{E}$. The following hold:*

(i) If $e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$, then $e' \in \rho(e)$.

(ii) If $e' \in \rho(e)$ and $e' \xrightarrow{\mathtt{a}} e''$, then $e'' \in \rho(e)$.

Proof. We treat the claims in the order given.

(i) Suppose $e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$. We prove that $e' \in \rho(e)$ by induction on e. In the base, where $e = 0$, $e = 1$ or $e = \mathtt{a}$ for some $\mathtt{a} \in \Sigma$, the claim holds by definition of $\to_{\mathbb{E}}$ and ρ. For the inductive step, there are three cases to consider.

- If $e = e_1 + e_2$, then $e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$ implies that either $e_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$ or $e_2 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$; we assume the former w.l.o.g. By induction, $e' \in \rho(e_1) \subseteq \rho(e)$.
- If $e = e_1 \cdot e_2$, then there are two more subcases.
  - If $e' = e'_1 \cdot e_2$ with $e_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'_1$, then by induction $e'_1 \in \rho(e_1)$. This implies that $e'_1 \in \rho(e_1)$ by induction, and hence $e' = e'_1 \cdot e_2 \in \rho(e)$.
  - If $e' = e'_2$ with $e_2 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'_2$, then by induction $e'_2 \in \rho(e_2) \subseteq \rho(e)$.
- If $e = e_1^*$, then $e' = e'_1 \cdot e_1^*$ with $e_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'_1$. By induction we have that $e'_1 \in \rho(e_1)$, and thus $e' = e'_1 \cdot e_1^* \in \rho(e)$.

(ii) Suppose $e' \in \rho(e)$ and $e' \xrightarrow{\mathtt{a}}_{\mathbb{E}} e''$. We show that $e'' \in \rho(e)$ by induction on e. In the base, the claim again holds immediately. For the inductive step, there are three cases to consider.

- If $e = e_1 + e_2$, then either $e' \in \rho(e_1)$ or $e' \in \rho(e_2)$; we assume the former without loss of generality. By induction, we then have that $e'' \in \rho(e_1)$, and hence $e'' \in \rho(e)$.
- If $e = e_1 \cdot e_2$, then there are two subcases to consider.
  - If $e' = e'_1 \cdot e_2$ with $e'_1 \in \rho(e_1)$, then either $e'' = e''_1 \cdot e_2$ with $e'_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e''_1$, or $e'' = e'_2$ with $e_2 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'_2$. In the former case, $e''_1 \in \rho(e_1)$ by induction, and hence $e'' = e''_1 \cdot e_2 \in \rho(e)$. In the latter case, $e'_2 \in \rho(e_2)$ by the first property of this lemma, and thus $e'' = e'_2 \in \rho(e)$ as well.
  - If $e' \in \rho(e_2)$, $e_1 \in \mathbb{A}$ and $e_2 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$, then $e' \in \rho(e_2) \subseteq \rho(e)$.
- If $e = e_1^*$, then $e' = e'_1 \cdot e^*$ with $e'_1 \in \rho(e_1)$. There are two subcases to consider.
  - If $e'' = e''_1 \cdot e^*$ with $e'_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e''_1$, then by induction $e''_1 \in \rho(e_1)$, and hence $e = e''_1 \cdot e^* \in \rho(e)$.
  - If $e'_1 \in \mathbb{A}$ and $e'' = e''_1 \cdot e_1^*$ where $e_1 \xrightarrow{\mathtt{a}}_{\mathbb{E}} e''_1$, then $e''_1 \in \rho(e_1)$ by the first property, and hence $e'' = e''_1 \cdot e_1^* \in \rho(e)$. □

We are now in a position to define our automaton for an expression e.

Definition 3.13. Let $e \in \mathbb{E}$, and write $\hat{\rho}(e)$ for $\rho(e) \cup \{e\}$. We write A_e for the *Antimirov automaton* of e, which is given by $\langle \hat{\rho}(e), \rightarrow_{\mathbb{E}}, \{e\}, \mathbb{A} \cap \hat{\rho}(e) \rangle$.

Note that since $\rho(e)$ is finite for every e, so is A_e.

Example 3.14. The reachable part of the Antimirov automaton for $\mathtt{a} \cdot \mathtt{b}^*$ is very similar to the automaton in Figure 5; the only difference is that the state on the right is labelled by $1 \cdot \mathtt{b}^*$, since $\mathtt{a} \cdot \mathtt{b}^* \xrightarrow{\mathtt{a}}_{\mathbb{E}} 1 \cdot \mathtt{b}^*$.

In the literature, when $e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'$ you may see e' referred to as a $\mathtt{a}$-*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 3.15 (Fundamental theorem)**.** *Let $e \in \mathbb{E}$. The following holds:*[2]

$$e \equiv [e \in \mathbb{A}] + \sum \{\mathtt{a} \cdot e' : e \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'\}$$

The proof proceeds by induction on e, and is part of today's homework. The fundamental theorem is very useful; for one thing, it lets us show quite easily that the Antimirov automaton for e accepts precisely the language of e as an expression, which we record in the following lemma.

[2]The generalized summation is a bit of an abuse of syntax, in two ways. First, it does not specify the order or bracketing of the sum; fortunately, the latter is not an issue when it comes to equivalence, by the axioms that we used to build $\equiv$. Second, and more unfortunately, the generalized $\sum$ overlaps with the traditional symbol Σ used to denote the alphabet; we persist in this notation for now, as it is always clear from context what we mean.

Lemma 3.16. *Let $e \in \mathbb{E}$. Now $L(A_e) = [\![e]\!]_{\mathbb{E}}$.*

Proof. Let $w \in \Sigma^*$. We prove that for all $e' \in \hat{\rho}(e)$, it holds that $w \in L(e')$ if and only if $w \in [\![e']\!]_{\mathbb{E}}$, by induction on w. In the base, $w = \epsilon$. By Theorem 3.15 and soundness, we find that $\epsilon \in [\![e']\!]_{\mathbb{E}}$ if and only if $e' \in \mathbb{A}$, i.e., $\epsilon \in L(e')$.

For the inductive step, let $w = \mathtt{a}w'$. By Theorem 3.15 and soundness, $w \in [\![e']\!]_{\mathbb{E}}$ if and only if $w' \in [\![e'']\!]_{\mathbb{E}}$ for some $e'' \in \mathbb{E}$ with $e' \xrightarrow{\mathtt{a}}_{\mathbb{E}} e''$. By induction, the latter is equivalent to $w' \in L(e'')$, and hence to $w = \mathtt{a}w' \in L(e')$. □

We can now take a step back, and look at what we have: given expressions e and f, we can construct automata A_e and A_f such that $L(A_e) = [\![e]\!]_{\mathbb{E}}$ and $L(A_f) = [\![f]\!]_{\mathbb{E}}$. We can then convert these automata into *deterministic* automata A'_e and A'_f such that $L(A'_e) = [\![e]\!]_{\mathbb{E}}$ and $L(A'_f) = [\![f]\!]_{\mathbb{E}}$. Given these deterministic automata, it is necessary and sufficient to find a bisimulation relating $\{e\}$ to $\{f\}$ to conclude that $[\![e]\!]_{\mathbb{E}} = [\![f]\!]_{\mathbb{E}}$. We return to this question in the next lecture.

7 Homework

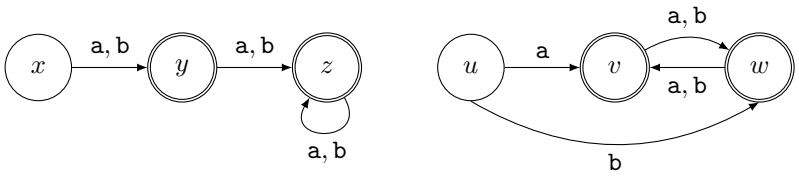
1. Create an automaton for each of the following languages over $\Sigma = \{\mathtt{a}, \mathtt{b}\}$:

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

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

2. *(Optional)* Try to write a regular expression whose language coincides with the language of the automaton on the right in Figure 1. Can you come up with a succinct (prosaic) description of the language?

3. Consider the automata drawn below:



   Prove that x and u are bisimilar.

4. *(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:[3]

   (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 $R \cup T \subseteq R'$.

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

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.

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

6. Prove the fundamental theorem (Theorem 3.15) 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

$$e_i \equiv [e_i \in \mathbb{A}] + \sum \{\mathtt{a} \cdot e' : e_i \xrightarrow{\mathtt{a}}_{\mathbb{E}} e'\}$$

   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 $(b + f)^* \equiv f^*$.

7. Use Antimirov's construction to come up with an automaton for the expression $\mathtt{b}^* \cdot \mathtt{a}$. Start from the expression, and expand the automaton from there; you do not need to include any unreachable states in $\rho(\mathtt{b}^* \cdot \mathtt{a})$.

8. Apply the powerset construction to the automaton obtained from the previous exercise. You may omit unreachable states once more.

9. *(Optional)* Construct the Antimirov automaton for $e = \mathtt{a} + \mathtt{a} \cdot \mathtt{b} + \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{b}^*$. Is the state e bisimilar to the state $\mathtt{a} \cdot \mathtt{b}^*$ in Figure 5?

8 Bibliographical notes

Automata developed over time, but some of the very first ideas appeared in the work of McCulloch and Pitts [MP43]. Rabin and Scott [RS59] first discussed the constrast between deterministic and non-deterministic automata, and proposed the powerset construction. In fact, the latter paper specifically won them the Turing award, sometimes called the "Nobel prize for computer science".

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 is a simplified version of a more sophisticated algorithm due to Hopcroft and Karp [HK71]. There is also a kind of efficient bisimulation that works for non-deterministic automata, due to Bonchi and Pous [BP13].

Kleene [Kle56] first observed the connection between rational expressions and automata. Antimirov's construction [Ant96] was based on Brzozowski's construction [Brz64], which produces a deterministic automaton instead (but is slightly more complicated). Later constructions include an inductive construction by Thompson [Tho68]. For a good overview, 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.

[BP13] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. Principles of Programming Languages (POPL)*, pages 457–468, 2013. doi:10.1145/2429069.2429124.

[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.

[RS59] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

[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.