DFA, NFA, AFW on finite words

Giuseppe De Giacomo
Sapienza Università di Roma

In this document we revisit the classical theory of finite state automata, having in mind as application the verification and synthesis of finite state processes from language-based specifications.

We start by considering an example from digital design [7].

Example 1 (A Simple Vending Machine).

- **Requirements.** *The vending machine delivers a package of gum after it has received 15 cents in coins. The machine has a single coin slot that accepts nickels (5c) and dimes (10c), one coin at a time. A mechanical sensor indicates to the control whether a dime or a nickel has been inserted into the coin slot. The controller's output causes a single package of gum to be released down a chute to the customer. One further specification: We will design our machine so it does not give change. A customer who pays with two dimes is out 5 cents.*

  We proceed to the manual synthesis of the machine in steps.

- **Solution step 1: Understanding the problem.** *The first step in the finite state machine design process is to understand the problem. Event* 5c *is asserted when a nickel is inserted into the coin slot. Event* 10c *is asserted when a dime has been deposited. The event gum is asserted when 15 cents (or more) has been deposited since the last reset. The specification may not completely define the behavior of the finite state machine. For example, what happens if someone inserts a penny (1c) into the coin slot? Or what happens after the gum is delivered to the customer? Sometimes we have to make reasonable assumptions. For the first question, we assume that the coin sensor returns any coins it does not recognize, leaving* 5c *and* 10c *unasserted. For the latter, we assume that the machine resets after the gum is delivered.*

- **Solution step 2: Define an abstract representation as a DFA.** *Once you understand the behavior reasonably well, it is time to map the specification into a more suitable abstract representation as a finite state machine. A good way to begin is by enumerating the possible unique sequences of inputs or configurations of the system. These will help define the states of the finite state machine.*

For this problem, it is not too difficult to enumerate all the possible input sequences that lead to releasing the gum:

- *three nickels in sequence:* 5*c*, 5*c*, 5*c*
- *two nickels followed by a dime:* 5*c*, 5*c*, 10*c*
- *a nickel followed by a dime:* 5*c*, 10*c*
- *a dime followed by a nickel:* 10*c*, 5*c*
- *two dimes in sequence:* 10*c*, 10*c*

This can be represented as a state diagram. For example, the machine will pass through the three states if the input sequence is three nickels.

To keep the state diagram simple and readable, we include only transitions that explicitly cause a state change. For example, in the initial state, if neither input 5*c or* 10*c is asserted, we assume the machine remains in the initial state (the specification allows us to assume that* 5*c and* 10*c are never asserted at the same time). Also, we include the event gum only in states in which it is asserted. The event gum is implicitly unasserted in any other state.*

- **Solution step 3: State minimization of the DFA.** *Typically the* DFA *resulting from the previous step isn't the "best" possible. States that have identical behavior can be combined into a single state. To reduce the number of states even further, we can think of each state as representing the amount of money received so far. For example, it shouldn't matter whether the state representing that 10c have been received was reached through two nickels or one dime. The process of minimizing the states in a finite state machine description is called* state minimization*.*

- **Solution step 4: State encoding and implementation.** *At this point, we have a finite state machine with a minimum number of states, but it is still symbolic. The next step is state encoding into actual flip-flop logic. The way you encode the state can have a major effect on the amount of hardware you need to implement the machine. For example, a natural state assignment would encode the states in 2 bits: state 0c received as* 00*, state 5c received as* 01*, state 10c received as* 10*, and state 15c received as* 11*. A less obvious assignment could lead to reduced hardware. The next step is to implement the state transition table on the basis of the chosen storage elements (various kinds of flip-flops).*

Suppose that two different teams come up with two different designs: the DFA A and the DFA A' in Figure 2.

- *How can we understand that they implement the same design?*

  A sufficient condition for this is that they recognize the same language. To check this, we can proceed as follows: we check weather $\mathcal{L}(A) \subseteq \mathcal{L}(A')$ and $\mathcal{L}(A') \subseteq \mathcal{L}(A)$. To do so, focussing on the first containment, we consider that $\mathcal{L}(A) \subseteq \mathcal{L}(A')$ iff $\mathcal{L}(A) \cap \overline{\mathcal{L}(A')} =$

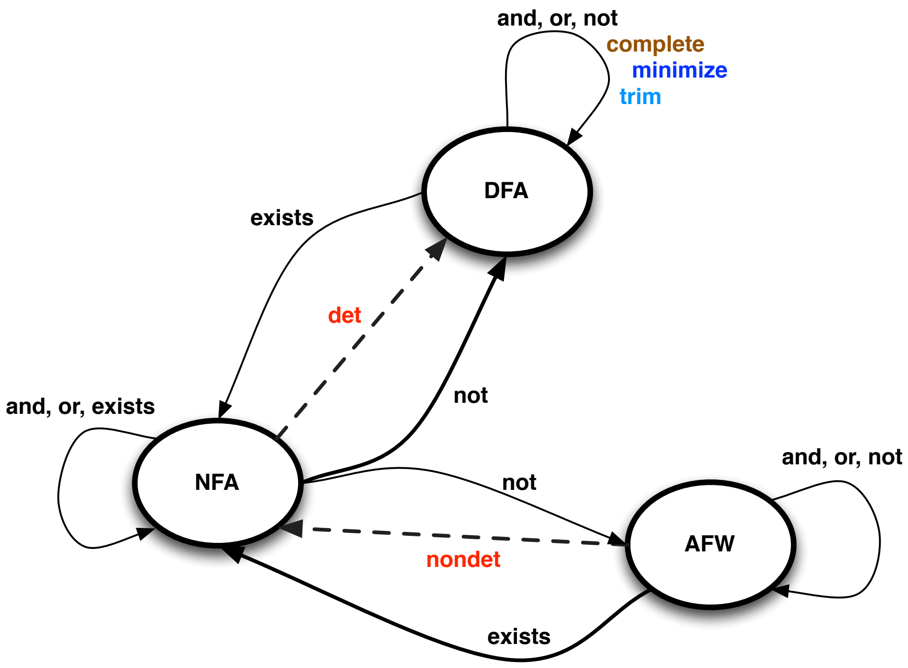

Figure 1: Relationship between DFA, NFA, and AFW

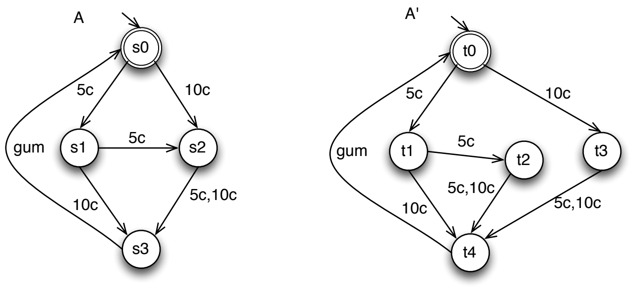

Figure 2: Two equivalent DFAs: A, which is minimal, and A', which is not minimal

$\emptyset$, which can be computed as $\mathcal{L}(A \wedge \overline{A'}) = \emptyset$, where $\overline{A'}$ is the DFA recognizing the complement of the language recognized by A', and $A \wedge \overline{A'}$ denotes the DFA recognizing the intersection of the languages recognized by A and $\overline{A'}$. Similarly for the other containment.

- *Are the two designs minimal?*

  Actually A is indeed minimal, while A' is not. If we apply minimization to A we get A', since for every DFA there is a single minimal equivalent DFA (modulo renaming of states).

In these notes we study and give automata theoretic tools for Steps 2 and 3 of the above synthesis methodology. Figure 1 summarizes what we present. In the figure, thin edges correspond to polynomial transformations, while thick ones correspond to transformations that induce an exponential blowup.

1 DFA: Deterministic Finite Automata

We are given a finite nonempty alphabet Σ . A finite word is an element of Σ^*, i.e., a finite sequence $a_0 \cdots a_n$ of symbols from Σ. Automata on finite words define (finitary) languages, i.e., sets of finite words.

We start our review by considering deterministic finite automata first. A DFA, *deterministic finite automaton*, A is a tuple $A = (\Sigma, S, s^0, \rho, F)$, where

- Σ is a finite nonempty *alphabet*;
- S is a finite nonempty set of *states*;
- $s^0 \in S$ is the *initial* state;
- $F \subseteq S$ is the set of *accepting* states;
- $\rho : S \times \Sigma \to S$ is a *transition function*, which can be a partial function. Intuitively, $s' = \rho(s, a)$ is the state that A can move into when it is in state s and it reads the symbol a. (If $\rho(s, a)$ is undefined then reading a leads to rejection.)

An automaton is essentially an edge-labeled directed graph: the states of the automaton are the nodes, the edges are labeled by symbols in Σ, a node is designated as initial, and a certain set of nodes is designated as accepting (or final). Thus, $s' = \rho(s, a)$ means that that there is an edge from s to s' labeled with a. Examples of DFAs are reported in Figure 2.

The transition function ρ of a DFA can be viewed as a partial mapping from $S \times \Sigma$ to S, and can then be extended to a partial mapping from $S \times \Sigma^*$ to S as follows:

- $\rho(s, \epsilon) = s$;
- $\rho(s, xw) = \rho(\rho(s, x), w)$, for $s \in S$, $x \in \Sigma$, and $w \in \Sigma^*$.

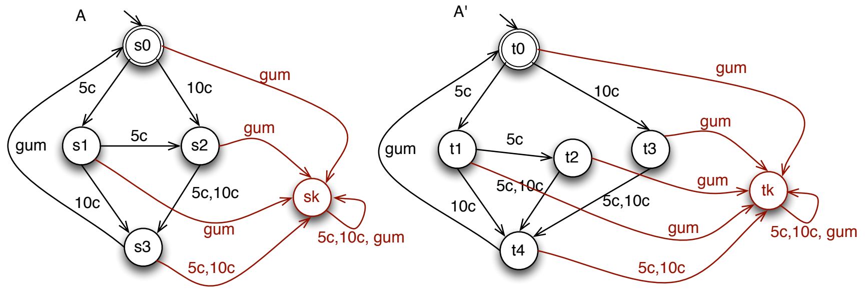

Figure 3: Completions of DFAs A and A'

A run r of A on a finite word $w = a_0 \cdots a_{n-1} \in \Sigma^*$ is a sequence $s_0 \cdots s_n$ of $n+1$ states in S such that $s_0 = s^0$, and $s_{i+1} = \rho(s_i, a_i)$ for $0 \leq i \leq n$. Note that a deterministic automaton can have at most one run on a given input word. The run r is *accepting* if $s_n \in F$. One could picture the automaton as having a green light that is switched on whenever the automaton is in an accepting state and switched off whenever the automaton is in a non-accepting state. Thus, the run is accepting if the green light is on at the end of the run. The word w is *accepted by* A if A has an accepting run on w. Since A is deterministic, $w \in \mathcal{L}(A)$ if and only if $\rho(s^0, w) \in F$.

The semantics of a DFA A is the (possibly infinite) set $\mathcal{L}(A)$ of finite words accepted by A. $\mathcal{L}(A)$ is typically called the *language of* A. Notice that DFAs are semantically fully characterized by the language that they recognize. Their structure, number of states, transitions defined for each state, etc. are irrelevant with respect to semantics.

One key observation about DFAs is that they can be easily implemented in software and in hardware: they correspond to actual physical/virtual releasable devices or machines. This has to be contrasted to nondeterministic and alternating automata to be introduced later, which do not correspond directly to implementable devices (with current technology).

1.1 Completion of a DFA

We say that a DFA is complete if its transition function $\rho : S \times \Sigma \to S$ is a total function, that is, for all $s \in S$ and all $a \in \Sigma$ we have that $\rho(s, a) = s'$ for some $s' \in S$ (i.e., $\rho(s, a)$ is defined).

Given an arbitrary DFA $A = (\Sigma, S, s^0, \rho, F)$, its completed version A_T, called *completion of* A, is immediately obtained as follows: $A_T = (\Sigma, S \cup \{sink\}, s_0, \rho_T, F)$ where:

$$\rho_T(s, a) = \begin{cases} \rho(s, a), & \text{if defined in } A \\ sink, & \text{otherwise} \end{cases}$$

That is, the completion A_T of A is obtained from A by adding an extra state *sink*, and by making the transition function ρ total by setting $\rho_T(s, a) = sink$ in A_T every time that $\rho(s, a)$ is undefined in A. Examples of completions are reported in Figure 3.

Theorem 1 (**DFA Completion** [11]). *Let A be a* DFA. *Its completion A_T is such that $\mathcal{L}(A) = \mathcal{L}(A_T)$. The size of A_T is linear in the size of A (its states are that of A plus one).*

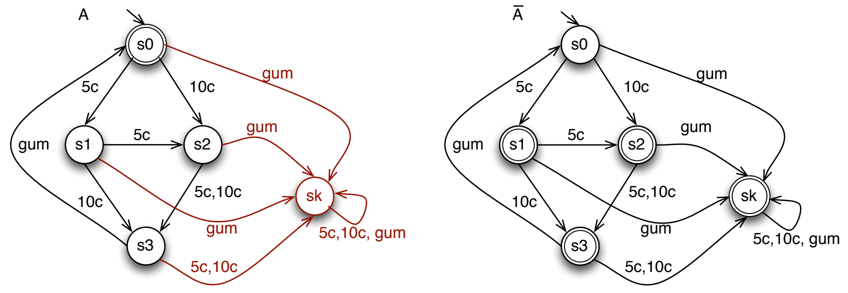

Figure 4: Complementation (not) of DFA A

Proof. By the construction above. $\square$

1.2 Closure of DFAs under Complementation, Intersection, and Union

An important property of automata is their closure under Boolean operations: complement, intersection, and union.

Closure under complementation (not). Let $A = (\Sigma, S, s^0, \rho, F)$. We assume that A is complete (if not, apply the completion construction above). Intuitively, we construct a DFA $\bar{A}$ that runs A but accepts whenever A does not. We define

$$\bar{A} = (\Sigma, S, s^0, \rho, S - F).$$

Notice that $\bar{A}$ does exactly the same transitions as the DFA A (which is complete, and hence transitions are always defined!), but accepts only the words that are not accepted by A. That is $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$. As an example, see Figure 4.

Theorem 2 (**DFA Closure Under Complementation** [11])**.** *Let A be DFA. Then there is a DFA $\bar{A}$ such that $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$. The size of $\bar{A}$ is linear in A.*

Proof. By construction above. $\square$

Closure under intersection (and). Let $A_1 = (\Sigma, S_1, s_1^0, \rho_1, F_1)$ and $A_2 = (\Sigma, S_2, s_2^0, \rho_2, F_2)$. Intuitively, we construct a DFA $A_\wedge$ that runs simultaneously both A_1 and A_2 on the input word and accepts when *both* accept. We define $A_\wedge$ as:

$$A_\wedge = (\Sigma,\ S_1 \times S_2,\ (s_1^0, s_2^0),\ \rho,\ F_1 \times F_2)$$

where:

- The transition function requires to execute the two automata concurrently in a synchronized way:

$$\rho((s_1, s_2), a) = (s_1', s_2') \quad \text{iff} \quad s_1' = \rho_1(s_1, a) \text{ and } s_2' = \rho_2(s_2, a)$$

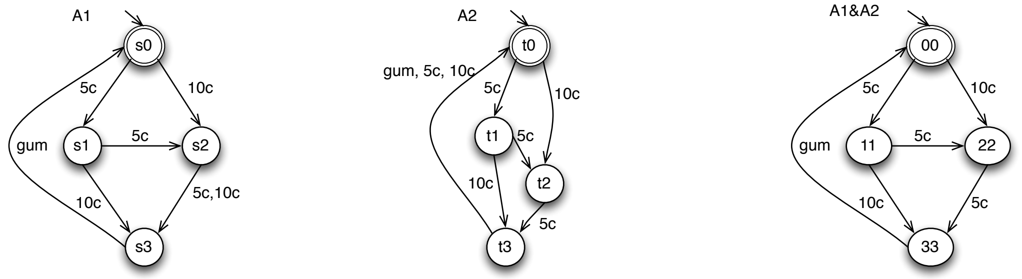

Figure 5: Intersection (and) of DFAs $A1$ and $A2$

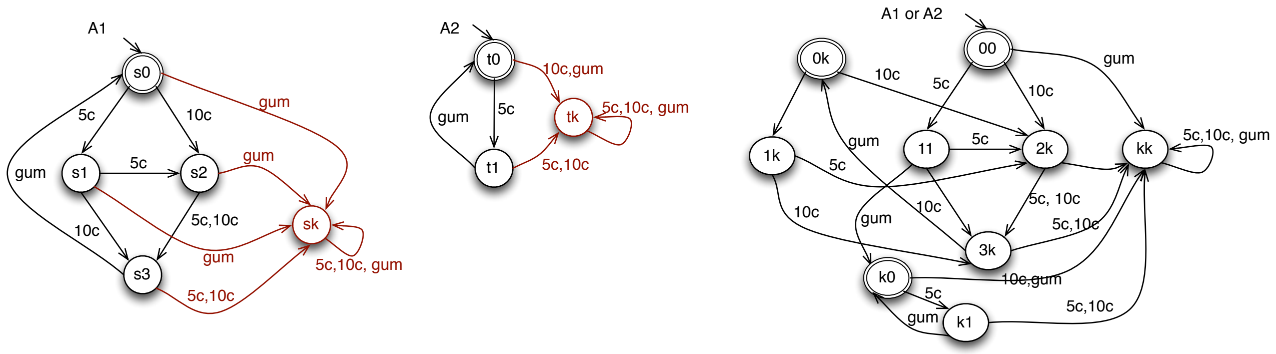

Figure 6: Union (or) of DFA's $A1$ and $A2$

Notice that if either $\rho_1(s_1, a)$ or $\rho_2(s_2, a)$ is undefined then also $\rho((s_1, s_2), a)$ is undefined.

- The condition defining the final states:

$$F_1 \times F_2$$

says that we accept a word if both DFAs A_1 and A_2 accept it.

It is easy to see that the size of $A_\wedge$ is the product of the sizes of A_1 and of A_2, and that $\mathcal{L}(A_\wedge) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. As an example, see Figure 5.

Theorem 3 (**DFA Closure Under Intersection** [11]). *Let A_1 and A_2 be* DFA*s. Then there is a* DFA *A such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. The size of A is linear in the product of the sizes of A_1 and of A_2.*

Proof. By the construction above. □

Sometimes $A_\wedge$ defined above is called the *product* of A_1 and A_2, and is denoted by $A_1 \times A_2$.

Closure under union (or). We observe that by De Morgan laws on Boolean operators, we have

$$\mathcal{L}(A_1) \cup \mathcal{L}(A_2) = \overline{\overline{\mathcal{L}(A_1)} \cap \overline{\mathcal{L}(A_2)}}$$

Here, however we give a direct construction for the union. Let $A_1 = (\Sigma, S_1, s_1^0, \rho_1, F_1)$ and $A_2 = (\Sigma, S_2, s_2^0, \rho_2, F_2)$. Without loss of generality, we assume that S_1 and S_2 are disjoint and that A_1 and A_2 are *complete* (i.e., their transition function is total, which is needed for the implicit complementation above). Intuitively, we construct a DFA $A_\vee$ that runs simultaneously both A_1 and A_2 on the input word and accepts when *one of them* accepts. Notice that the two DFAs can never get suck (with no transition available), since they are complete. We define

$$A_\vee = (\Sigma,\ S_1 \times S_2,\ (s_1^0, s_2^0),\ \rho,\ (F_1 \times S_2) \cup (S_1 \times F_2))$$

where:

- The transition function requires to execute the two automata concurrently in a synchronized way:

  $$\rho((s_1, s_2), a) = (s_1', s_2') \quad \text{iff} \quad s_1' = \rho_1(s_1, a) \text{ and } s_2' = \rho(s_2, a)$$

  Though we require that both DFAs A_1 and A_2 are complete and hence $\rho_1(s_1, a)$ and $\rho_2(s_2, a)$ are always both defined.

- The condition defining the final states:

  $$(F_1 \times S_2) \cup (S_1 \times F_2)$$

  says that $A_\vee$ accepts a word if one of the two DFAs A_1 or A_2 accepts it.

It is easy to see that the size of $A_\vee$ is the product of the sizes of A_1 and of A_2, and that $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. As an example see Figure 6.

Theorem 4 (**DFA Closure Under Union** [11])**.** *Let A_1 and A_2 be* DFA*s. Then there is a* DFA *A such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. The size of A is polynomial in sizes of A_1 and of A_2.*

Proof. By the construction above. □

Sometimes $A_\vee$ defined above is called the *union* of A_1 and A_2, and is denoted as $A_1 \cup A_2$.

1.3 Projection of DFAs

We define the operation of "projecting out the subset X of the alphabet Σ" as the operation that removes from a word all occurrence of symbols in X. In other words, projecting out X from a word corresponds to the word obtained by existentially quantifying over the symbols in X.

Projection $\pi_X(w)$ wrt X of a word w is defined by induction on the length of w as follows:

- $\pi_X(\epsilon) = \epsilon$
- $\pi_X(aw) = \begin{cases} aw, & \text{if } a \in \Sigma - X \\ w, & \text{if } a \in X \end{cases}$

Given a language L, the projection $\pi_X(L)$ is the language

$$\pi_X(L) = \{w' \mid \exists w \in L \ s.t. \ w' = \pi_X(w)\}.$$

Given a DFA $A = (\Sigma, S, s^0, \rho, F)$, we can define an NFA A_{π_X} that recognizes the language $\pi_X(\mathcal{L}(A))$.

To do so, first we define the relation $\varepsilon_X \subseteq S \times S$ formed by the pairs of states (s, s') such that s' is reachable from s by making transition involving only symbols from X, as the smallest relation ε_X such that:

- $(s, s) \in \varepsilon_X$
- if $(s, s') \in \varepsilon_X$ then for all s'' such that $s'' = \delta(s', a)$ for some $a \in X$, we have that $(s, s'') \in \varepsilon_X$.

Then we define A_{π_X} as follows:

$$A_{\pi_X} = (\Sigma - X, S, S_0, \rho_X, F)$$

where:

- $S_0 = \{s \mid (s^0, s) \in \varepsilon_X\}$
- $(s, a, s') \in \rho_X$ iff there exist t, t' s.t. $(s, t) \in \varepsilon_X$, $t' = \rho(t, a)$ and $(t', s') \in \varepsilon_X$

Theorem 5 (DFA Projection). *Given a* DFA *A, there exists an* NFA *A_{π_X} computable in quadratic time such that $\mathcal{L}(A_{\pi_X}) = \pi_X(\mathcal{L}(A))$.*

Proof. The NFA A_{π_X} is the one defined above. To build it we only need to compute reachability between states, which can be done in quadratic time. □

1.4 Structural Operations on DFAs

In this section we present some common structural operations that change the structure of a DFA without changing the language that the DFA recognizes.

1.4.1 Minimization

Given a DFA $A = (\Sigma, S, s_0, \rho, F)$ there exists a single minimal DFA A_m which is equivalent to A, i.e., $\mathcal{L}(A) = \mathcal{L}(A_m)$ and with a minimal number of states.

To construct such a DFA we need to introduce a suitable equivalence relation between states.

We exploit bisimulation[1] between states of the DFA A, which we assume to be *complete* (if not we first apply the completion operation introducing a *sink* state, see above).

A bisimulation relation $\mathcal{E} \in S \times S$ is a relation between states that satisfies the following condition: if $(s,t) \in \mathcal{E}$ then:

- $s \in F$ iff $t \in F$;
- For all s', a such that $\rho(s,a) = s'$, there exists t' such that $\rho(t,a) = t'$ and $(s',t') \in \mathcal{E}$;
- For all t', a such that $\rho(t,a) = t'$, there exists s' such that $\rho(s,a) = s'$ and $(s',t') \in \mathcal{E}$.

We say that s is *equivalent to* t, written $s \sim t$ if there exists a bisimulation relation $\mathcal{E}$ such that $(s,t) \in \mathcal{E}$. It is to be shown that $s \sim t$ is indeed an equivalence relation: i.e., it is:

- reflexive (i.e., $s \sim s$);
- symmetric (i.e., if $s \sim t$ then $t \sim s$);
- transitive (i.e., if $s \sim t$ and $t \sim q$ then $s \sim q$).

Notice that this is a coinductive definition which can be computed as a greatest fixpoint as follows:

- We set $Z_0 = S \times S$.
- Then, we from Z_i we compute Z_{i+1} as:

  $$Z_{i+1} = Z_i - \{(s,t) \mid (s,t) \text{ does } \textbf{not} \text{ satisfy one of the three conditions below}\}$$

  – $s \in F$ iff $t \in F$ (in fact this is used only when going from Z_0 to Z_1);
  – for all s', a s.t. $\rho(s,a) = s'$, there exists t' s.t. $\rho(t,a) = t'$ and $(s',t') \in Z_i$;
  – for all t', a s.t. $\rho(t,a) = t'$, there exists s' s.t. $\rho(s,a) = s'$ and $(s',t') \in Z_i$.

- When for some n we get $Z_{n+1} = Z_n$, then we set $s \sim t$ iff $(s,t) \in Z_n$.

Once we have such an equivalence relation, we choose one state s for each equivalence class $[s] = \{t \mid s \sim t\}$, including s^0 for the equivalence class $[s^0]$. Then we define the minimal DFA $A_m = (\Sigma, S_m, s^0, \rho_m, F_m)$ as follows:

[1]Notice that we should look at trace equivalence instead which is coarser that bisimilarity, however the two notions coincide for DFAs.

	t0	t1	t2	t3	t4	tk
t0	▨	░	░	░	░	░
t1	x	▨	░	░	░	░
t2	x	x	▨	░	░	░
t3	x	x		▨	░	░
t4	x	x	x	x	▨	░
tk	x	x	x	x	x	▨

Figure 7: Conflict matrix for the DFA A' of Figure 3.

- S_m is formed by the chosen states s (one for each equivalence class $[s]$);
- $\rho_m(s, a) = s'$ if $\rho(s, a) = s''$ and $s' \in [s''] \cap S_m$;
- $F_m = \{s \mid s \in F \cap S_m\}$.

Notice that there is only one such minimal DFA modulo renaming of the states. This minimal DFA A_m can be considered as the *canonical form* of A, which is indeed unique (modulo renaming of states).

Interestingly there is a svelte way of computing the equivalence relation among states.

1. We build a matrix formed by the states of the original (complete) DFA A both in the columns and in the rows.
2. Since the equivalence relation is symmetric and reflexive we can consider only the part of the matrix that is below the diagonal.
3. Now we apply the fixpoint algorithm by marking all those pairs of states (elements in the matrix) for which we have a conflict. Namely:
   - We mark all pairs such that one is final and the other is not.
   - Then we mark those pairs of states such that for some symbol in Σ we go to a pair of states that is already marked.
   - We repeat the marking step until no more violation can be found.
4. The pairs of states that are not marked are equivalent.

Applying this methods to the DFA A' in Figure 3, we get the conflicts matrix in Figure 7. From such a conflict matrix we learn that $t2 \sim t3$, so we can merge the two states getting the DFA A of Figure 3.

1.4.2 Trimming

Given a DFA $A = (\Sigma, S, s^0, \rho, F)$, we describe some handy structural modifications of A that do not impact the recognized language but structurally simplify the DFA.

Reachable DFA. We can remove from A all *unreachable states* as follows: We define the set $S_R \subset S$ of states that are *reachable from the initial state* s^0 by induction as the smallest S_R set such that:

- $s^0 \in S_R$;
- if $s \in S_R$, then for every s' such that $s' = \rho(s, a)$ for some $a \in \Sigma$, we have that $s' \in S_R$.

Then we define the *reachable* DFA A_R corresponding to A as

$$A_R = (\Sigma,\, S_R,\, s^0,\, \rho|_{S_R},\, F \cap S_R)$$

where $\rho|_{S_R}$ is the restriction on $S_R \times \Sigma$ of ρ.

Co-reachable DFA. Similarly we can remove from A all states that *do not reach a final state.* We define the set $S_F \subseteq S$ of states that reach a state in F by induction as the smallest set such that:

- $F \subseteq S_F$;
- if $s' \in S_F$, then for every s such that $\rho(s, a) = s'$ for some $a \in \Sigma$, we have that $s \in S_F$.

Then, assuming that $s^0 \in S_F$ (if not the resulting DFA is empty), we define the *co-reachable* DFA A_F corresponding to A as

$$A_F = (\Sigma,\, S_F,\, s^0,\, \rho|_{S_F},\, F)$$

where $\rho|_{S_F}$ is the restriction on $S_F \times \Sigma$ of ρ.

Trimmed DFA. Finally we define the *trimmed* DFA A_{RF} corresponding to A as the DFA

$$A_{RF} = (\Sigma,\, S_R \cap S_F,\, s^0,\, \rho|_{S_R \cap S_F},\, F \cap S_R)$$

where $\rho|_{S_R \cap S_F}$ is the restriction on $(S_R \cap S_F) \times \Sigma$ of ρ. Notice that the trimmed DFA contains only those states that are reachable from the initial state and that lead to a final state.

Theorem 6. *Let A be a* DFA *and A_R its corresponding reachable* DFA*, A_F its corresponding co-reachable* DFA*, and A_{RF} its corresponding trimmed* DFA*. Then $\mathcal{L}(A) = \mathcal{L}(A_R) = \mathcal{L}(A_F) = \mathcal{L}(A_{RF})$ and computing A_R, A_F, and A_{RF} requires linear time in the size of A.*

Observe that the trimmed version of the minimal DFA corresponding to A (see above) is the most compact DFA recognizing the language $\mathcal{L}(A)$.

2 NFA: Nondeterministic Finite Automata

An NFA, *nondeterministic finite automaton*, A is a tuple $A = (\Sigma, S, S^0, \rho, F)$, where

- Σ is a finite nonempty *alphabet*;
- S is a finite nonempty set of *states*;
- S^0 is the nonempty set of *initial* states;
- F is the set of *accepting* states;
- $\rho : S \times \Sigma \times S$ is a *transition relation.* Intuitively, $(s, a, s') \in \rho$ states that A can move from s into s' when it reads the symbol a. It is allowed that $(s, a, s') \in \rho$ and $(s, a, s'') \in \rho$ with $s' \neq s''$.

It is also possible to define ρ as a function $\rho : S \times \Sigma \to 2^S$ returning sets of states:

$$\rho(s, a) = \{s' \mid (s, a, s') \in \rho\}$$

Note that NFAs are generally nondeterministic, since they have many initial states and the transition relation may specify several possible transitions for each state and symbol. The automaton A is *deterministic* if $|S^0| = 1$ and for all s, a, s', s'' we have that $(s, a, s') \in \rho$ and $(s, a, s'') \in \rho$ implies $s' = s''$ (or, alternatively, $|\rho(s, a)| = 1$, for all s and a).

A run r of A on a finite word $w = a_0 \cdots a_{n-1} \in \Sigma^*$ is a sequences $s_0 \cdots s_n$ of $n+1$ states in S such that $s_0 \in S^0$, and $(s_i, a_i, s_{i+1}) \in \rho$ for $0 \leq i \leq n$. Note that a nondeterministic automaton can have many runs on a given input word. In contrast, a deterministic automaton can have at most one run on a given input word. The run r is *accepting* if $s_n \in F$. The word w is *accepted by* A if there *exists* an accepting run of A on w.

The semantics of an NFA A is the (possibly infinite) set $\mathcal{L}(A)$ of finite words accepted by A, i.e., the language, accepted by A. Notice that, as DFAs, also NFAs are semantically fully characterized by the language that they recognize, while their structure, number of states, transitions defined for choices of actions at each state, etc. are irrelevant with respect to semantics.

2.1 Closure of NFAs under Intersection and Union

Nondeterministic automata are closed under Intersection and Union, as DFAs.

Closure under intersection (and). Let $A_1 = (\Sigma, S_1, S_1^0, \rho_1, F_1)$ and $A_2 = (\Sigma, S_2, S_2^0, \rho_2, F_2)$ be two NFAs. Intuitively, we construct an NFA $A_\wedge$ that runs simultaneously both A_1 and A_2 on the input word.

Let $A_\wedge = (\Sigma, S, S^0, \rho, F)$ where:

- $S = S_1 \times S_2$

- $S^0 = S_1^0 \times S_2^0$
- $F = F_1 \times F_2$
- $((s,t), a, (s',t')) \in \rho$ iff $(s,a,s') \in \rho_1$ and $(t,a,t') \in \rho_2$

It is easy to see that $\mathcal{L}(A_\wedge) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

Theorem 7 (NFA Closure Under Intersection [11]). *Let A_1 and A_2 be two* NFA*s. Then there is a* NFA *A such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. The size of A is polynomial in the sizes of A_1 and of A_2.*

Proof. By the construction above. □

As in the case of DFAs, we call $A_\wedge$ in the proof above the *product* of A_1 and A_2, denoted $A_1 \times A_2$.

Closure under union (or). Let $A_1 = (\Sigma, S_1, S_1^0, \rho_1, F_1)$ and $A_2 = (\Sigma, S_2, S_2^0, \rho_2, F_2)$ be two NFAs. Without loss of generality, we assume that S_1 and S_2 are disjoint. Intuitively, we construct an NFA $A_\vee$ that nondeterministically chooses A_1 or A_2 and runs it on the input word.

Let $A_\vee = (\Sigma, S, S^0, \rho, F)$ where:

- $S = S_1 \cup S_2$
- $S^0 = S_1^0 \cup S_2^0$
- $F = F_1 \cup F_2$
- $\rho = \rho_1 \cup \rho_2$, that is $(s,a,s') \in \rho = \begin{cases} s \in S_1 \text{ and } (s,a,s') \in \rho_1, \text{ or} \\ s \in S_2 \text{ and } (s,a,s') \in \rho_2 \end{cases}$

It is easy to see that $\mathcal{L}_\vee(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

Theorem 8 (NFA Closure Under Union [11]). *Let A_1 and A_2 be two* NFA*s. Then there is an* NFA *A such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. The size of A is polynomial in the sizes of A_1 and of A_2.*

Proof. By the construction above. □

As in the case of DFAs, we call $A_\vee$ in the proof above the *union* of A_1 and A_2, denoted $A_1 \cup A_2$.

Note that both the union and the product constructions are effective and polynomial in the sizes of the constituent automata.

2.2 Closure of NFAs under Complementation and Determination

Let us consider now the issue of complementation. As seen for Theorem 2, it is easy to complement deterministic automata; we just have to complement the acceptance condition. This will not work for nondeterministic automata, since a nondeterministic automaton can have many runs on a given input word; it is not enough that some of these runs reject (i.e., not accept) the input word, *all runs should reject* the input word. Thus, it seems that to complement a nondeterministic automaton we first have to *determinize* it.

Determinization. Let $A = (\Sigma, S, S^0, \rho, F)$ be an NFA. Then we can define a DFA A_d such that $\mathcal{L}(A_d) = \mathcal{L}(A)$ as $A_d = (\Sigma, 2^S, s^0, \rho_d, F_d)$ where:

- 2^S, i.e., the state set of A_d, consists of all sets of states S in A;
- $s^0 = S^0$, i.e., the single initial state of A_d is the set S^0 of initial states of A;
- $F_d = \{Q \mid Q \cap F \neq \emptyset\}$, i.e., the collection of sets of states that intersect F nontrivially;
- $\rho_d(Q, a) = \{s' \mid (s, a, s') \in \rho \text{ for some } s \in Q\}$.[2]

Intuitively, A_d collapses all possible runs of A on a given input word into one run over a larger state set. This construction is called the *subset construction.*

Theorem 9 (NFA to DFA [11]). *Let A be an NFA. Then there exists a DFA A_d such that $\mathcal{L}(A_d) = \mathcal{L}(A)$.*

Proof. By the construction above. We show that each string w belongs to $\mathcal{L}(A)$ iff $\mathcal{L}(A_d)$, by induction on the length of the string w. □

By combining Theorems 9 and 2, we can complement a nondeterministic automaton. The construction is effective, but it involves an exponential blow-up, since determinization involves an exponential blow-up (i.e., if A has n states, then A_d has 2^n states). As shown in [9], this exponential blow-up for determinization and complementation is unavoidable.

Example 2. The following example [13] shows that the exponential blow-up for determinization is unavoidable. Fix some $n \geq 1$. The set of all finite words over the alphabet $\Sigma = \{a, b\}$ that have an a at the n-th position from the right is accepted by the automaton $A = (\Sigma, \{0, 1, 2, \ldots, n\}, \{0\}, \rho, \{n\})$ where $\rho(0, a) = \{0, 1\}$, $\rho(0, b) = \{0\}$, and $\rho(i, a) = \rho(i, b) = \{i + 1\}$ for $1 \leq i < n$. Intuitively, A guesses a position in the input word, checks that it contains a, and then checks that it is at distance n from the right end of the input. Suppose that we have a deterministic automaton $A_d = (\Sigma, S, s^0, \rho_d, F)$ with fewer than 2^n states that accepts this same language. Recall that ρ_d can be viewed as a partial mapping from $S \times \Sigma^*$ to S. Since $|S| < 2^n$ there must be two distinct words uav_1 and ubv_2 of length n for which $\rho_d(s^0, uav_1) = \rho_d(s^0, ubv_2)$ (i.e., u is the longest common prefix of the two words). But then we would have that $\rho_d(s^0, uav_1u) = \rho_d(s^0, ubv_2u)$, that is, either

[2]Note that $\rho_d(Q, a)$ may be the empty set (of states of A), which is one of the states of the DFA A_d.

both uav_1u and ubv_2u are members of $\mathcal{L}(A_d)$ or neither are. Since $|av_1u| = |bv_2u| = n$, this contradicts the assumption that $\mathcal{L}(A_d)$ consists of exactly the words with an a at the n-th position from the right. □

2.3 Nonemptiness

An automaton is "interesting" if it defines an "interesting" language, i.e., a language that is neither empty nor contains all possible words. An automaton A is *nonempty* if $\mathcal{L}(A) \neq \emptyset$; it is *nonuniversal* if $\mathcal{L}(A) \neq \Sigma^*$. One of the most fundamental algorithmic issues in automata theory is testing whether a given automaton is "interesting", i.e., nonempty and nonuniversal. The *nonemptiness problem* for automata is to decide, given an automaton A, whether A is nonempty. The *nonuniversality problem* for automata is to decide, given an automaton A, whether A is nonuniversal. It turns out that for NFAs testing nonemptiness is easy, while testing nonuniversality is hard.

Theorem 10 (**NFA Nonemptiness** [11, 6]).
The nonemptiness problem for NFA*s is decidable in linear time, and is* NLogSpace-*complete.*

Proof. Let $A = (\Sigma, S, S^0, \rho, F)$ be an NFA. Let s, s' be states in S. We say that s' is *directly connected* to s if there is a symbol $a \in \Sigma$ such that $(s, a, s') \in \rho$. We say that s' is *connected* to s if there is a sequence $s_1, \ldots, s_m$, with $m \geq 1$, of states such that $s_1 = s$, $s_m = s'$ and s_{i+1} is directly connected to s_i for $1 \leq i < m$. Essentially, t is connected to s if there is a path in A from s to t, where A is viewed as an edge-labeled directed graph. Note that the edge labels are ignored in this definition. It is easy to see that $\mathcal{L}(A)$ is nonempty iff there are states $s \in S^0$ and $t \in F$ such that t is connected to s. Thus, automata nonemptiness is equivalent to graph reachability. The claims now follow from the following observations:

1. A breadth-first-search algorithm can construct in linear time the set of all states connected to a state in S^0 [3]. A is nonempty iff this set intersects F nontrivially.

2. Graph reachability can be tested in *nondeterministic* logarithmic space. The algorithm simply guesses a state $s_0 \in S^0$, then guesses a state s_1 that is directly connected to s_0, then guesses a state s_2 that is directly connected to s_1, etc. When it reaches a state in F, it accepts. (Recall that a nondeterministic algorithm accepts if there is a sequence of guesses that leads to acceptance. We do not care here about sequences of guesses that do not lead to acceptance [4].) In addition, the algorithm keeps a counter with which it can count up to the number n of states. If after n steps the algorithm does not reach a state in F, then it rejects. At each step the algorithm needs to remember only the current state, the next state, and the value of the counter; thus, with n states the algorithm needs to keep in memory $O(\log n)$ bits, since $3 \cdot \log n$ bits suffice to describe two states and to store the value of the counter. On the other hand, graph reachability is also NLogSpace-hard [6]. □

Theorem 11 (NFA Nonuniversality [11, 6]**).**
The nonuniversality problem for NFA*s is decidable in exponential time, and is* PSpace*-complete.*

Proof. Note that $\mathcal{L}(A) \neq \Sigma^*$ iff $\Sigma^* - \mathcal{L}(A) \neq \emptyset$ iff $\mathcal{L}(\bar{A}) \neq \emptyset$, where $\bar{A}$ is the complementary automaton of A. Thus, to test A for nonuniversality, it suffices to test $\bar{A}$ for nonemptiness. Recall that $\bar{A}$ is exponentially bigger than A. Since nonemptiness can be tested in linear time, it follows that nonuniversality can be tested in exponential time. Also, since nonemptiness can be tested in nondeterministic logarithmic space, nonuniversality can be tested in polynomial space.

The latter argument requires some care. We cannot simply construct $\bar{A}$ and then test it for nonemptiness, since $\bar{A}$ is exponentially big. Instead, we construct $\bar{A}$ "on-the-fly": whenever the nonemptiness algorithm wants to move from a state t_1 of $\bar{A}$ to a state t_2, the algorithm guesses t_2 and checks that it is directly connected to t_1. Once this has been verified, the algorithm can discard t_1. Thus, at each step the algorithm needs to keep in memory at most two states of $\bar{A}$ (i.e., two sets of states of A, if we consider $\bar{A}$ obtained from A via the subset construction) and there is no need to generate all of $\bar{A}$ at any single step of the algorithm.

This yields a nondeterministic polynomial space algorithm. To eliminate nondeterminism, we appeal to a well-known theorem of Savitch [12] which states that $\text{NSpace}(f(n)) \subseteq \text{DSpace}(f(n)^2)$, for $f(n) \geq \log n$; that is, any nondeterministic algorithm that uses at least logarithmic space can be simulated by a deterministic algorithm that uses at most a quadratically larger amount of space. In particular, any nondeterministic polynomial-space algorithm can be simulated by a deterministic polynomial-space algorithm.

To prove PSpace-hardness, it can be shown that any PSpace-hard problem can be reduced to the nonuniversality problem. That is, there is a logarithmic-space algorithm that given a polynomial-space-bounded Turing machine M and a word w, outputs an automaton $A_{M,w}$ such that M accepts w iff $A_{M,w}$ is non-universal [10, 5]. $\square$

3 AFW: Alternating Finite Automata on Words

Nondeterminism gives a computing device the power of existential choice. Its dual gives a computing device the power of universal choice. (Compare this to the complexity classes NP and coNP [4]). It is therefore natural to consider computing devices that have the power of both existential choice and universal choice. Such devices are called alternating. Alternation was studied in [2] in the context of Turing machines and in [1, 2] for finite automata. The alternation formalisms in [1] and [2] are different, though equivalent. We follow here the formalism of [1].

For a given set X , let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas *true* and *false*. Let $Y \subseteq X$ .We say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X - Y$ satisfies

θ. For example, the sets $\{s_1, s_3\}$ and $\{s_1, s_4\}$ both satisfy the formula $(s_1 \vee s_2) \wedge (s_3 \vee s_4)$, while the set $\{s_1, s_2\}$ does not satisfy this formula.

Consider an NFA $A = (\Sigma, S, S^0, \rho, F)$. The transition function ρ maps a state $s \in S$ and an input symbol $a \in \Sigma$ to a set of states. Each element in this set is a possible nondeterministic choice for the automatons next state. We can represent ρ by using $\mathcal{B}^+(S)$; for example, $\rho(s, a) = \{s_1, s_2, s_3\}$ can be written as $\rho(s, a) = s_1 \vee s_2 \vee s_3$. In alternating automata, $\rho(s, a)$ can be an arbitrary formula from $\mathcal{B}^+(S)$. We can have, for instance, a transition

$$\rho(s, a) = (s_1 \wedge s_2) \vee (s_3 \wedge s_4)$$

meaning that the automaton accepts the word aw, where a is a symbol and w is a word, when it is in the state s, if it accepts the word w from both s_1 and s_2 or from both s_3 and s_4. Thus, such a transition combines the features of existential choice (the disjunction in the formula) and universal choice (the conjunctions in the formula).

Formally, an AFW (*alternating finite automaton on words*) is a tuple $A = (\Sigma, S, s^0, \rho, F)$, where:

- Σ is a finite nonempty alphabet;
- S is a finite nonempty set of states;
- $s^0 \in S$ is the initial state (notice that, as in DFAs, we have a unique initial state);
- $F \subseteq S$ is the set of accepting states;
- $\rho : S \times \Sigma \to \mathcal{B}^+(S)$ is a transition function.

Because of the universal choice in alternating transitions, a run of an alternating automaton is a tree rather than a sequence.

A *tree* is a (finite or infinite) connected directed graph, with one node designated as the *root* and denoted by ϵ, and in which every non-root node has a unique parent (s is the *parent* of t and t is a *child* of s if there is an edge from s to t) and the root ϵ has no parent. The level of a node x, denoted $|x|$, is its distance from the root ϵ; in particular, $|\epsilon| = 0$. A *branch* $\beta = x_0, x_1, \ldots$ of a tree is a maximal sequence of nodes such that x_0 is the root ϵ and x_i is the parent of x_{i+1} for all $i \geq 0$. Note that β can be finite or infinite. A Σ-*labeled tree*, for a finite alphabet Σ, is a pair $(\tau, \mathcal{T})$, where τ is a tree and $\mathcal{T}$ is a mapping from $nodes(\tau)$ to Σ that assigns to every node of τ a label in Σ. We often refer to $\mathcal{T}$ as the labeled tree. A branch $\beta = x_0, x_1, \ldots$ of $\mathcal{T}$ defines an infinite word $\mathcal{T}(\beta) = \mathcal{T}(x_0), \mathcal{T}(x_1), \ldots$ consisting of the sequence of labels along the branch.

Formally, a *run* of A on a finite word $w = a_0 a_1 \cdots a_n$ is a finite S-labeled tree r such that:

- $r(\epsilon) = s^0$
- if $|x| = i < n$, $r(x) = s$ and $\rho(s, a_i) = \theta$, then x has k children $x_1, \ldots, x_k$, for some $k \leq |S|$, and $\{r(x_1), \ldots, r(x_k)\}$ satisfies θ.

For example, if $\rho(s^0, a) = (s_1 \wedge s_2) \vee (s_3 \wedge s_4)$ then the nodes of the run tree at level 1 include the label s_1 or the label s_2 and also include the label s_3 or the label s_4. Note that the depth of r (i.e., the maximal level of a node in r) is at most n, but not all branches need to reach such depth, since if $\rho(r(x), a_i) = \mathit{true}$, then x does not need to have any children. On the other hand, if $|x| = i < n$, then we cannot have $\rho(r(x), a_i) = \mathit{false}$, since *false* is not satisfiable.

The run tree r is *accepting* if all nodes at depth n are labeled by states in F. Thus, a branch in an accepting run has to hit the *true* transition or hit an accepting state after reading all the input word.

3.1 AFW and NFA

What is the relationship between alternating automata and nondeterministic automata? It turns out that just as nondeterministic automata have the same expressive power as deterministic automata but they are exponentially more succinct, alternating automata have the same expressive power as nondeterministic automata but they are exponentially more succinct.

Translating NFAs to AFWs. Let $A = (\Sigma, S, S^0, \rho, F)$ be an NFA. Then we define the AFW A_A such that $\mathcal{L}(A_A) = \mathcal{L}(A)$ as follows $A_A = (\Sigma, S \cup \{s^0\}, s^0, \rho_A, F)$ where s^0 is a new state and ρ_A is defined as follows:

- $\rho_A(s, a) = \bigvee_{(s,a,s') \in \rho} s'$, for all $a \in \Sigma$ and $s \in S$
- $\rho_A(s^0, a) = \bigvee_{s \in S^0, (s,a,s') \in \rho} s'$, for all $a \in \Sigma$

(We take an empty disjunction in the definition of A_A to be equivalent to *false*.) Essentially, the transitions of A are viewed as disjunctions in A_A. A special treatment is needed for the initial state, since we allow a set of initial states in nondeterministic automata, but only a single initial state in alternating automata.

Theorem 12 (**NFA to AFW** [1, 2, 8]). *Let A be* NFA. *Then there exists a* AFW *A_A such that $\mathcal{L}(A_A) = \mathcal{L}(A)$.*

Proof. By the construction above. □

Note that A_A has essentially the same size as A; that is, the descriptions of A_A and A have the same length.

Translating AFWs to NFAs. We now show that alternating automata are not more expressive than nondeterministic automata.

Let $A = (\Sigma, S, s^0, \rho, F)$ be an AFW. Then we define the NFA A_N such that $\mathcal{L}(A_N) = \mathcal{L}(A)$ as follows $A_N = (\Sigma, S_N, S_N^0, \rho_N, F_N)$ where:

- $S_M = 2^S$

- $S_N^0 = \{\{s^0\}\}$
- $F_N = 2^F$
- $(Q, a, Q') \in \rho_N$ iff Q' satisfies $\bigwedge_{s \in Q} \rho(s, a)$

Note that we take an empty conjunction in the definition of ρ_N to be equivalent to *true*; thus, $(\emptyset, a, \emptyset) \in \rho_N$.

Intuitively, A_N guesses a run tree of A. At a given point of a run of A_N, it keeps in its memory a whole level of the run tree of A. As it reads the next input symbol, it guesses the next level of the run tree of A.

Theorem 13 (**AFW to NFA** [1, 2, 8])**.** *Let A be an* AFW*. Then there exists an* NFA *A_N such that $\mathcal{L}(A_N) = \mathcal{L}(A)$.*

Proof. By the construction above. We show that each string w belongs to $\mathcal{L}(A)$ iff $\mathcal{L}(A_N)$, by induction on the length of the string w. □

The translation from AFWs to NFAs involves an exponential blow-up. As shown in [1, 2, 8], this blow-up is unavoidable.

Example 3. This example shows that the blow-up is unavoidable. Fix some $n \geq 1$, and let $\Sigma = \{a, b\}$. Let Γ_n be the set of all words that have two different symbols at distance n from each other. That is,

$$\Gamma_n = \{uavbw \mid u, w \in \Sigma^* \text{ and } v \in \Sigma^{n-1}\} \cup \{ubvaw \mid u, w \in \Sigma^* \text{ and } v \in \Sigma^{n-1}\}$$

It is easy to see that Γ_n is accepted by the NFA $A = (\Sigma, \{p, q\} \cup (\{1, \ldots, n\} \times \Sigma), \{p\}, \rho, \{q\})$ where:

- $\rho(p, a) = \{p, (1, a)\}$
- $\rho(p, b) = \{p, (1, b)\}$
- $\rho((i, a), x) = \{(i{+}1, a)\}$, for $x \in \Sigma$ and $0 < i < n$
- $\rho((i, b), x) = \{(i{+}1, b)\}$, for $x \in \Sigma$ and $0 < i < n$
- $\rho((n, a), a) = \emptyset$
- $\rho((n, a), b) = \{q\}$
- $\rho((n, b), b) = \emptyset$
- $\rho((n, b), a) = \{q\}$
- $\rho(q, x) = \{q\}$ for $x \in \Sigma$

Intuitively, A guesses a position in the input word, reads the input symbol at that position, moves n positions to the right, and checks that it contains a different symbol. Note that A has $2n + 2$ states. By Theorems 12 and 14 (see later), there is an AFW A_a with $2n + 3$ states that accepts the complementary language $\overline{\Gamma_n} = \Sigma^* - \Gamma_n$. Consider a nondeterministic automaton $A_{nd} = (\Sigma, S, S^0, \rho_{nd}, F)$ that accepts $\overline{\Gamma_n}$. Thus, A_{nd} accepts all words ww, where $w \in \Sigma^n$. For such a word ww, let $s_w^0 \cdots s_w^{2n}$ be an accepting run of A_{nd} on ww. Suppose now that A_{nd} has fewer than 2^n states. Then there are two distinct words $u, v \in \Sigma^n$ such that $s_u^n = s_v^n$. Thus, $s_u^0 \cdots s_u^n s_v^{n+1} \cdots s_v^{2n}$ is an accepting run of A_{nd} on uv, but $uv \notin \overline{\Gamma_n}$ since it has two different symbols at distance n from each other. □

3.2 Boolean Closure

One advantage of alternating automata is that, not only it is easy to take unions and intersection, but it is also easy to complement them. We first need to define the dual operation on formulas in $\mathcal{B}^+(X)$. Intuitively, the dual $\bar{\theta}$ of a formula θ is obtained from θ by switching $\vee$ and $\wedge$, and by switching *true* and *false*. For example, $\overline{x \vee (y \wedge x)} = x \wedge (y \vee x)$. (Note that we are considering formulas in $\mathcal{B}^+(X)$, so we cannot simply apply negation to these formulas.) Formally, we define the dual operation as follows:

- $\overline{x} = x$
- $\overline{true} = false$
- $\overline{false} = true$
- $\overline{\alpha \wedge \beta} = \overline{\alpha} \vee \overline{\beta}$
- $\overline{\alpha \vee \beta} = \overline{\alpha} \wedge \overline{\beta}$

Theorem 14 (**AFW Complementation** [1, 2, 8])**.** *Let A be* AFW*. Then there exists an* AFW *$\bar{A}$ such that $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$.*

Proof. Let $A = (\Sigma, S, s^0, \rho, F)$. Define $\bar{A} = (\Sigma, S, s^0, \bar{\rho}, S - F)$, where $\bar{\rho}(s, a) = \overline{\rho(s, a)}$ for all $s \in S$ and $a \in \Sigma$. That is, $\bar{\rho}$ is the dualized transition function. It can be shown that $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$. □

3.3 Nonemptiness

By combining Theorems 10 and 13, we can obtain a nonemptiness test for AFW.

Theorem 15 (**AFW Nonemptiness** [2])**.** *The nonemptiness problem (as well as the nonuniversality problem) for* AFW*s is decidable in exponential time, and is* PSPACE*-complete.*

Proof. All that remains to be shown is the PSpace-hardness of nonemptiness. Recall that PSpace-hardness of nonuniversality of NFAs was shown in Theorem 11 by a generic reduction. That is, there is a logarithmic-space algorithm that, given a polynomial-space bounded Turing machine M and a word w, outputs an NFA $A_{M,w}$ such that M accepts w iff $A_{M,w}$ is nonuniversal. By Theorem 12, there is an AFW A_a such that $\mathcal{L}(A_a) = \mathcal{L}(A_{M,w})$ and A_a has the same size as $A_{M,w}$. By Theorem 14, $\mathcal{L}(\overline{A_a}) = \Sigma^* - \mathcal{L}(A_a)$. Thus, $A_{M,w}$ is nonuniversal iff $\overline{A_a}$ is nonempty. Moreover, $\overline{A_a}$ has the same size as A_a, from which the claim follows. $\square$

Acknowledgements

We thank Diego Calvanese for the help revising an earlier version of this manuscript, and Caterina Marano for her notes on the lectures on Declarative Business Processes.

References

[1] J. A. Brzozowski and E. Leiss. Finite automata and sequential networks. *Theor. Comput. Sci.*, 10:19–35, 1980.

[2] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability — A guide to NP-completeness.* W. H. Freeman and Company, San Francisco (CA, USA), 1979.

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

[6] N. D. Jones. Space-bounded reducibility among combinatorial problems. *JCSS*, 11:68–75, 1975.

[7] R. H. Katz. *Contemporary Logic Design.* Benjamin-Cummings Publishing Co., Inc., 1994.

[8] E. L. Leiss. Succint representation of regular languages by boolean automata. *Theor. Comput. Sci.*, 13:323–330, 1981.

[9] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *SWAT (FOCS)*, pages 188–191, 1971.

[10] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. of the 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972.

[11] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[12] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

[13] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.