

DFA, NFA, AFW on finite words

Giuseppe De Giacomo

Università degli Studi di Roma "La Sapienza"
Roma, Italy



Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words



Introduction

In this deck of slides 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 [5].

Example (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.

Solution

We proceed to the manual synthesis of the machine in steps from digital design [5].

- **Solution step 1: Understanding the problem.**
- **Solution step 2: Define an abstract representation as a DFA.**
- **Solution step 3: State minimization of the DFA.**
- **Solution step 4: State encoding and implementation.**

Solution step 1: Understanding the problem

Example (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

Example (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: $5c, 5c, 5c$
- two nickels followed by a dime: $5c, 5c, 10c$
- a nickel followed by a dime: $5c, 10c$
- a dime followed by a nickel: $10c, 5c$
- two dimes in sequence: $10c, 10c$

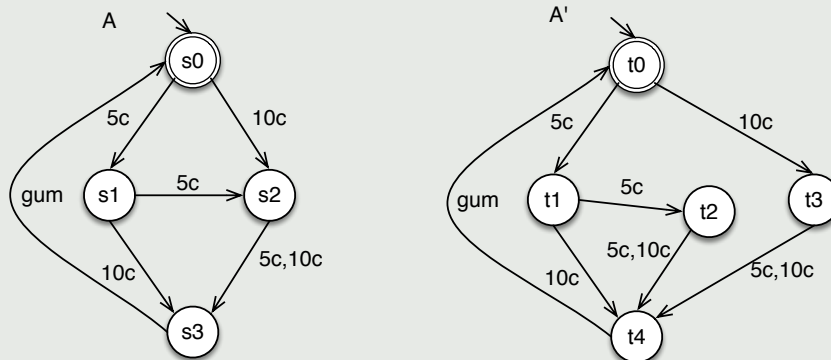
This can be represented as a state diagram. For example, the machine will pass through the three states if the input sequence is three nickels.

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 $5c$ or $10c$ is asserted, we assume the machine remains in the initial state (the specification allows us to assume that $5c$ and $10c$ 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 2: Define an abstract representation as a DFA - cont.d

Example (Solution step 2: Define an abstract representation as a DFA)

Two equivalent DFAs: A (which is minimal) and A' (which is not minimal)

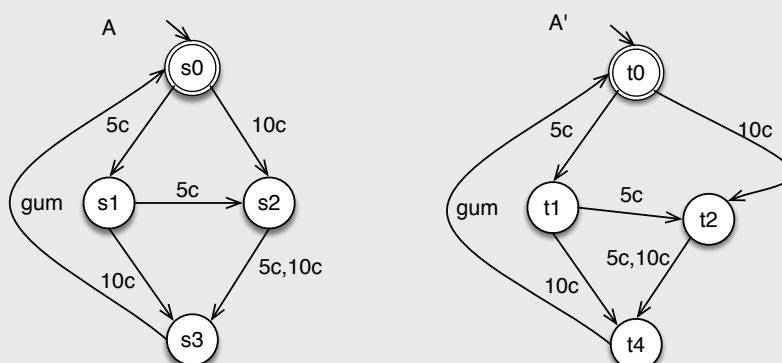


Solution step 3: State minimization of the DFA

Example (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**.

After minimization the two DFAs: A and A' become **identical**:



Solution step 4: State encoding and implementation

Example (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).

This step is not of direct interest for us.

Comments

Suppose that two different teams come up with two different designs a DFA A and a DFA A' as above.

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

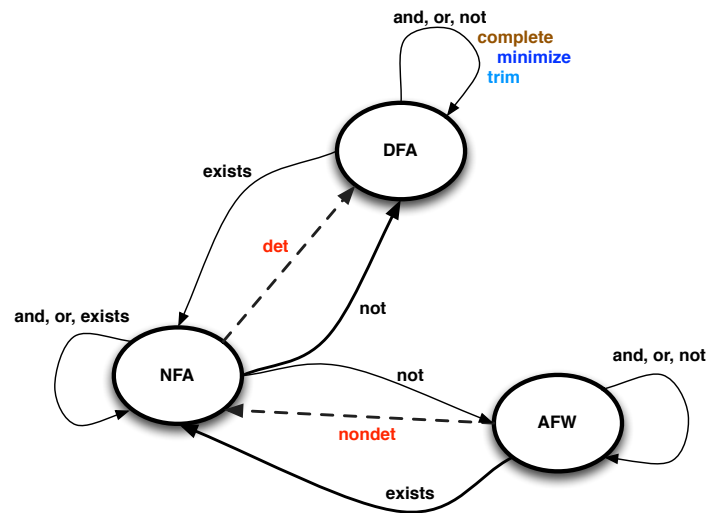
A sufficient condition for this is that they recognize the same language. To check this, we can proceed as follows: we check whether $\mathcal{L}(A) \subseteq \mathcal{L}(A')$ and $\mathcal{L}(A') \subseteq \mathcal{L}(A)$. To do so, focussing on the first containment, we consider that $\mathcal{L}(A)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

What we will study

In these notes we study and give automata theoretic tools for Steps 2 and 3 of the above synthesis methodology. The following figure summarizes what we will study.



In the figure, thin edges correspond to polynomial transformations, while thick ones correspond to transformations that induce an exponential blowup.

Outline

- 1 Introduction
- 2 **DFA: Deterministic Finite Automata**
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

DFAs: 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.

DFA

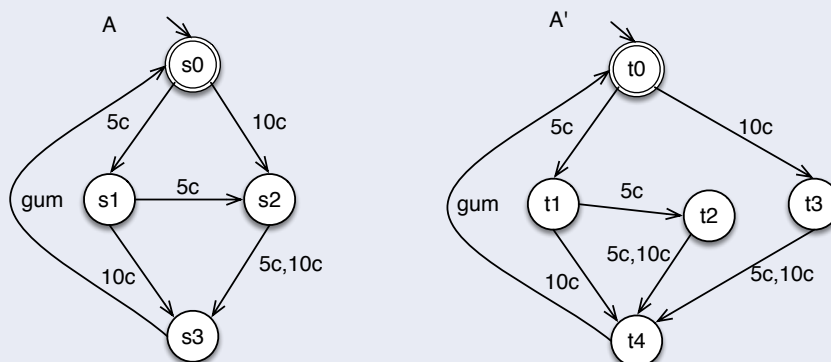
A DFA, **deterministic finite automaton**, A is a tuple $A = (\Sigma, S, s^0, \rho, F)$, where

- 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;
- $\rho : S \times \Sigma \rightarrow S$ 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.)

DFAs: Deterministic Finite Automata

DFA as edge-labeled directed graphs

An **automaton** is essentially an **edge-labeled directed graph**: the **states** of the automaton are the **nodes**, **transitions** are **edges**; 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 there is an edge from s to s' labeled with a .



DFAs: Deterministic Finite Automata

Extended transition function

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 $w \in \Sigma^*$.

Runs

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 $s_n \in F$. 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 $\rho(s^0, w) \in F$.

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.

 UNIVERSITA DI ROMA

Semantics

Semantics of a DFA

The **semantics** of a DFA A is the (possibly infinite) set 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).

Completion of a DFA

We say that a DFA is **complete** if its transition function $\rho : S \times \Sigma \rightarrow S$ 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).

Completion

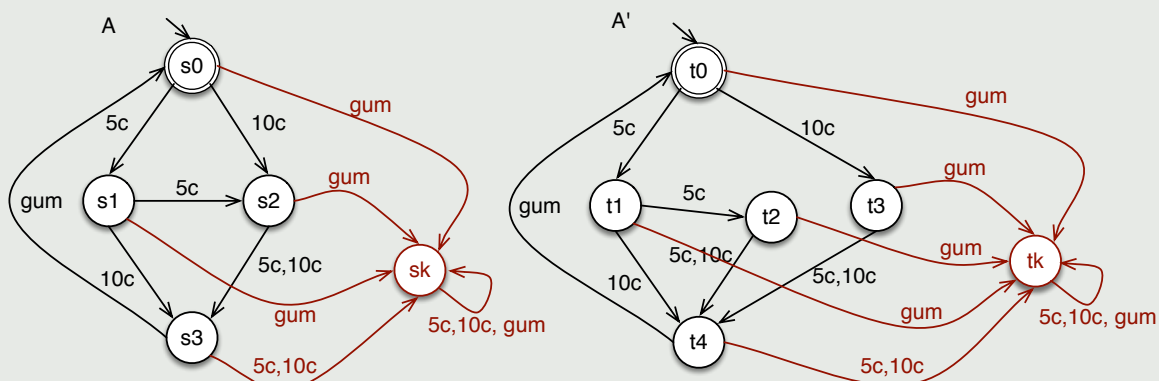
Given an arbitrary DFA $A = (\Sigma, S, s^0, \rho, F)$, its completed version A_T , called **completion of A** , is immediately obtained as follows: $A_T = (\Sigma, S \cup \{sink\}, s_0, \rho_T, F)$ where:

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

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 .

Completion of a DFA

Example (Completions of DFAs A and A')



Theorem (DFA Completion [7])

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).

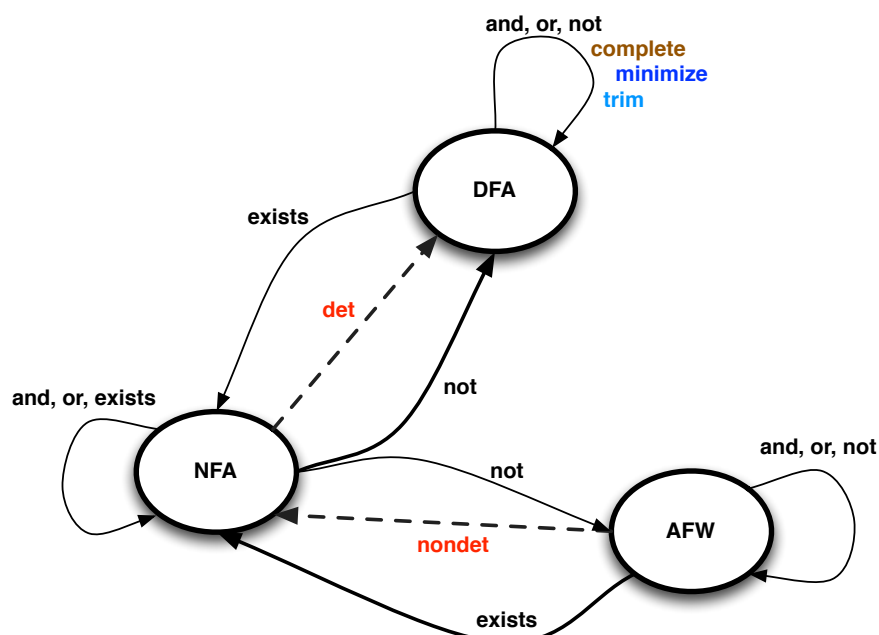
Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

Closure of DFAs under Complementation, Intersection, and Union

An important property of automata is their **closure under Boolean operations**:

- **complement (not)**,
- **intersection (and)**,
- **union (or)**



Closure under complementation (not)

Complementation

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. Formally:

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

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

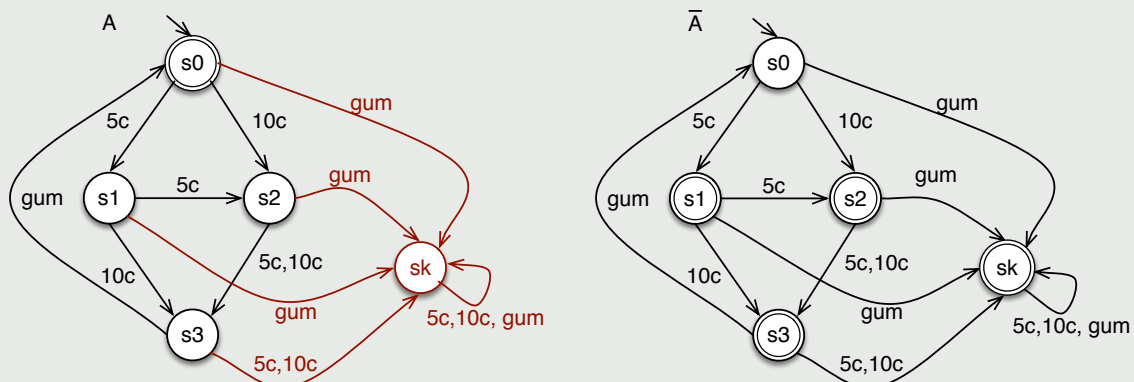
$$\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A).$$

Theorem (DFA Closure Under Complementation [7])

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 .

Closure under complementation (not)

Example (Complementation (not) of DFA A)



Closure under Intersection (and)

Intersection (and) of DFAs

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 the **intersection (and) 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:

$$\rho((s_1, s_2), a) = (s'_1, s'_2) \quad \text{iff} \quad s'_1 = \rho_1(s_1, a) \text{ and } s'_2 = \rho_2(s_2, a)$$

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.

Closure under Intersection (and)

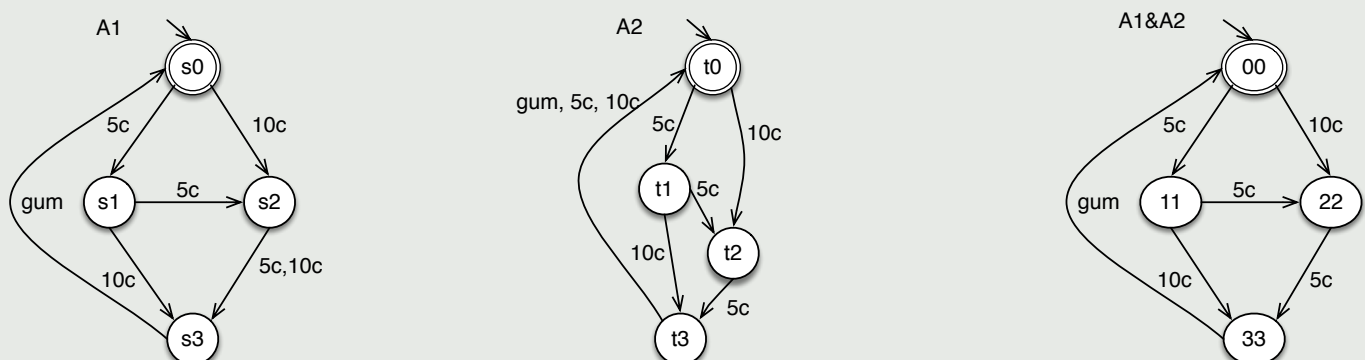
It is easy to see that the size of A_\wedge is the product of the sizes of A_1 and of A_2 , and that $\mathcal{L}(A_\wedge) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

Theorem (DFA Closure Under Intersection [7])

Let A_1 and A_2 be DFAs. 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 .

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

Example (Intersection (and) of DFAs A_1 and 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.

Union (or) of DFAs

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 the **union (or) DFA** A_V 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 stuck (with no transition available), since they are complete. We define

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

where:

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

$$\rho((s_1, s_2), a) = (s'_1, s'_2) \quad \text{iff} \quad s'_1 = \rho_1(s_1, a) \text{ and } s'_2 = \rho_2(s_2, a)$$

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_V accepts a word if one of the two DFAs A_1 or A_2 accepts it.

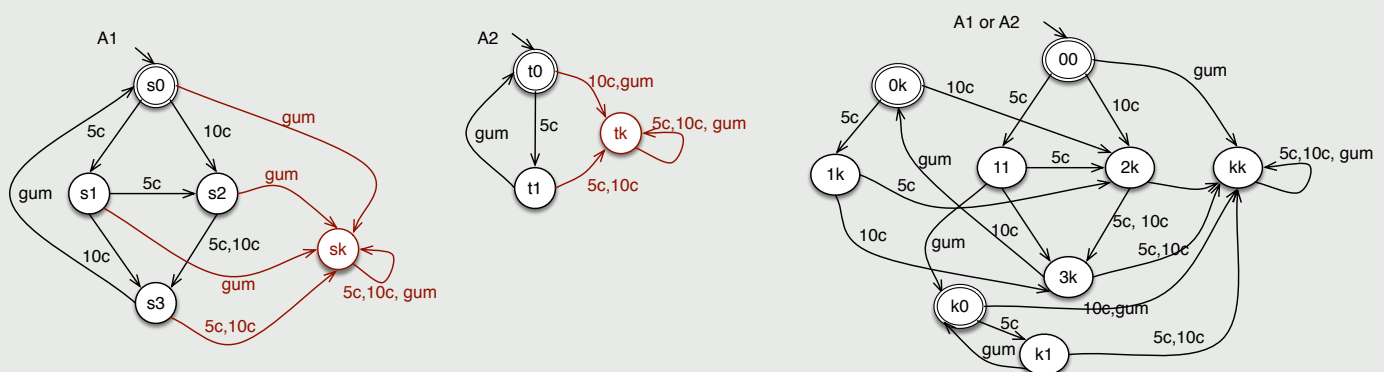
Closure under union (or)

It is easy to see that the size of A_V is the product of the sizes of A_1 and of A_2 , and that $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. Sometimes A_V defined above is denoted as $A_1 \cup A_2$.

Theorem (DFA Closure Under Union [7])

Let A_1 and A_2 be DFAs. 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 .

Example (Union (or) of DFA's A_1 and A_2)



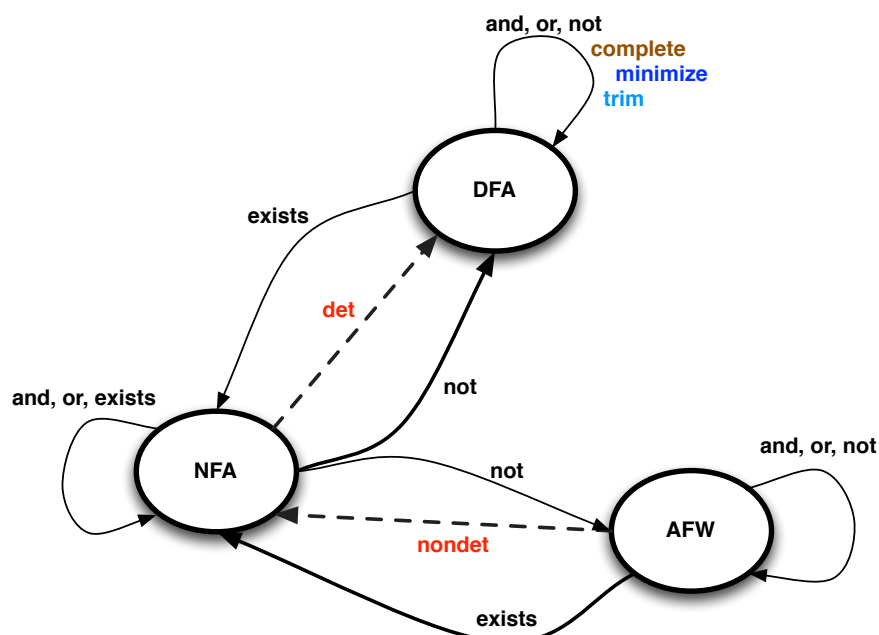
Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

Projection of DFAs

Appart from boolean operations automata are also closed under **existential quantification**, or **projection**, on part of the alphabet.

However if we apply existential quantification/projection to a DFA we get and NFA.



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 of a language

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

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

Projection of DFAs

Projection of a DFA

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

To do so, first we define by induction 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 the **projection NFA 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 (DFA Projection)

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

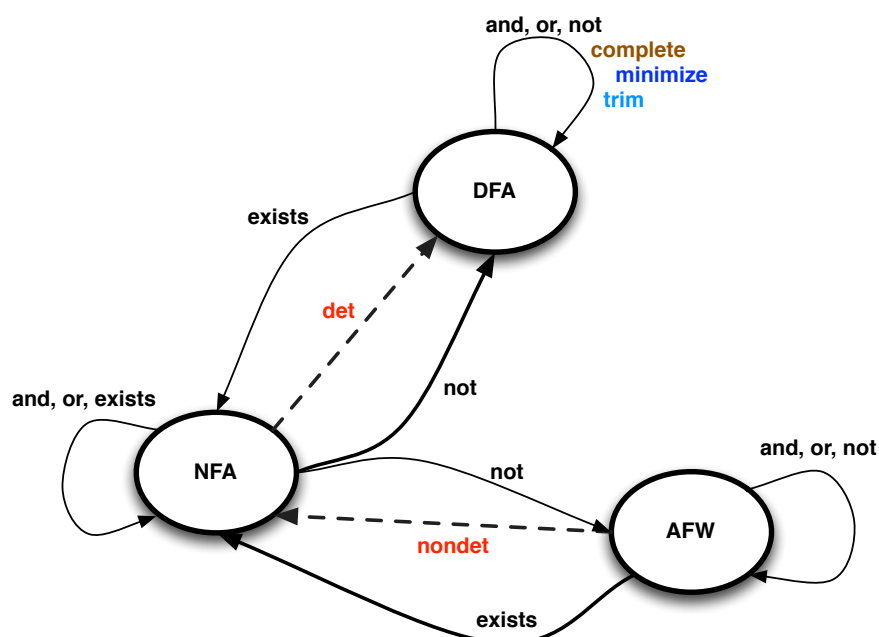
Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

Projection of DFAs

Now we present some common **structural operations** that change the structure of a DFA without changing the language that the DFA recognizes.

- Minimization
- Trimming



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.

Bisimulation

We exploit bisimulation^a 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 $s \sim q$).

^aNotice that we should look at trace equivalence instead which is coarser than bisimilarity, however the two notions coincide for DFAs.

Minimization

Computing Bisimulation

Notice that bisimulation 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:

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

- ▶ $s \in F$ 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 $(s, t) \in Z_n$.

Minimization

Minimal DFA

Once we have computed the (maximal) equivalence relation based on bisimulation, we choose one state s for each **equivalence class**:

$$[s] = \{t \mid s \sim t\}$$

We use 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:

- 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).

Minimization

Computing minimization

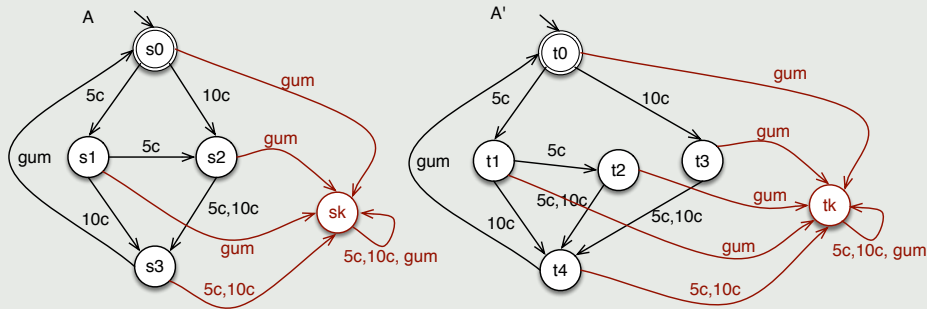
Interestingly there is a **svelte way of computing the (maximal) equivalence relation** among states.

- 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**.

Minimization

Example (Example of minimization)

Consider the DFA A'



We get the conflicts matrix:

t0						
t1	x					
t2	x	x				
t3	x	x				
t4	x	x	x	x		
tk	x	x	x	x	x	
	t0	t1	t2	t3	t4	tk

From such a conflict matrix we learn that $t2 \sim t3$, so we can merge the two states getting the DFA A of the figure.

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 .

Trimming

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 .

Trimming

Trimmed DFA

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

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 $\mathcal{L}(A)$.

Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

NFAs: Nondeterministic Finite Automata

NFA

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 $\rho : S \times \Sigma \rightarrow 2^S$ 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).*

NFAs: Nondeterministic Finite Automata

Runs

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 $s_0 \in S^0$, 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 .

Semantics

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 .

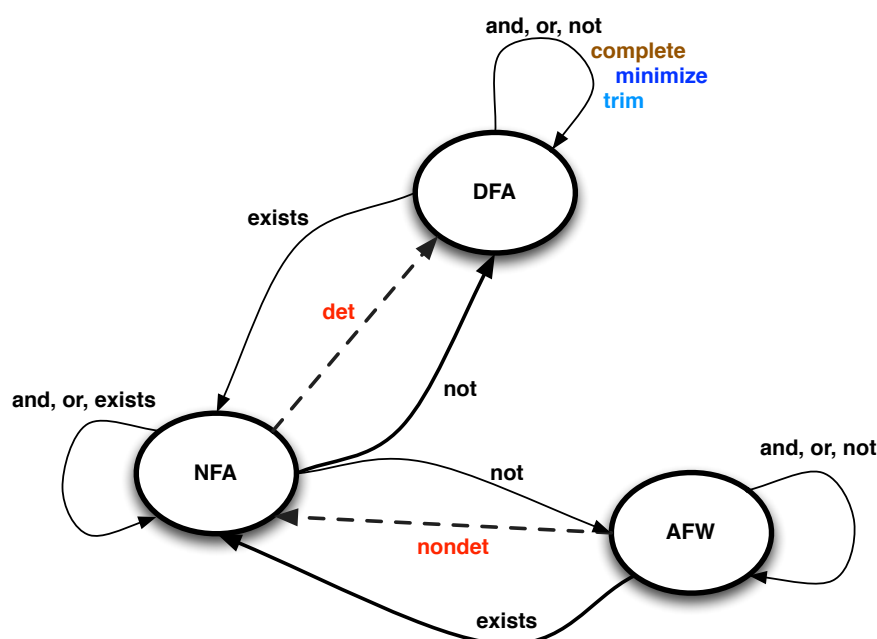
Notice that, as DFAs, also NFAs are semantically fully characterized by the language that they recognize, while their structure, number of states, transitions defined for choices of actions at each state, etc. are irrelevant with respect to semantics.



Closure of NFAs under Intersection and Union

Nondeterministic automata are closed under:

- **Intersection (and)**
- **Union (or)**



Closure under intersection (and)

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 (NFA Closure Under Intersection [7])

Let A_1 and A_2 be two NFAs. 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 .

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)

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 the **union 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 (NFA Closure Under Union [7])

Let A_1 and A_2 be two NFAs. 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 .

As in the case of DFAs, we denote the **union** A_\vee of A_1 and A_2 as $A_1 \cup A_2$.



Closure of NFAs under Complementation and Deteminization

Both the union and the product constructions are effective and polynomial in the sizes of the constituent automata. Let us consider now the issue of **complementation**.

We know that it is easy to complement deterministic automata; we just have to **complement the acceptance condition**.

This does 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, to complement a nondeterministic automaton we first have to **determinize** it, and then **complement to the resulting DFA**.

Determinization

Subset construction for 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;
- $\rho_d(Q, a) = \{s' \mid (s, a, s') \in \rho \text{ for some } s \in Q\}$.^a

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**.

^aNote 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 .

Theorem (NFA to DFA [7])

Let A be an NFA. Then there exists a DFA A_d such that $\mathcal{L}(A_d) = \mathcal{L}(A)$. The size of A_d can be exponential in A .

Nonemptiness and Nonuniversality

An automaton is “interesting” if it defines an “interesting” language, i.e., a language that is neither empty nor contains all possible words.

- **Nonemptiness**: an automaton A is **nonempty** if $\mathcal{L}(A) \neq \emptyset$;
- **Nonuniversality**: an automaton A is **nonuniversal** if $\mathcal{L}(A) \neq \Sigma^*$.

It turns out that for NFAs testing nonemptiness is easy, while testing nonuniversality is hard.

Theorem (NFA Nonemptiness [7, 4])

The nonemptiness problem for NFAs is decidable in linear time, and is NLOGSPACE-complete.

How to check: check reachability of a final state.

Theorem (NFA Nonuniversality [7, 4])

The nonuniversality problem for NFAs is decidable in exponential time, and is PSPACE-complete.

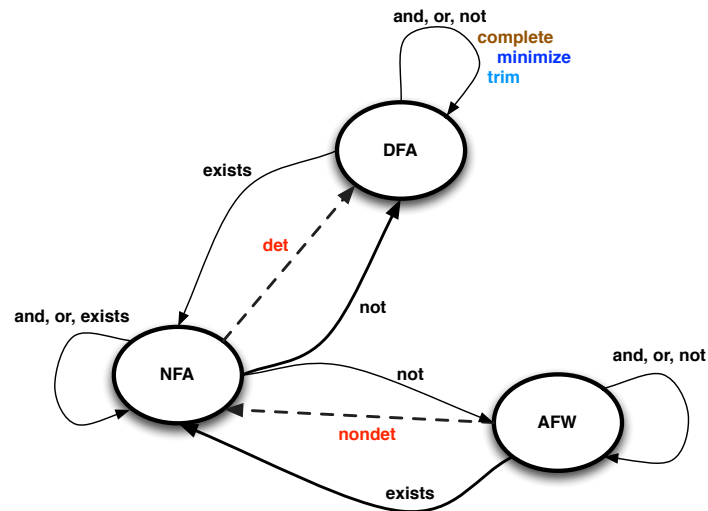
How to check: Determinize, complement and check for nonemptiness (on-the-fly).

Outline

- 1 Introduction
- 2 DFAs: Deterministic Finite Automata
- 3 Boolean closure of DFAs
- 4 Projection of automata
- 5 Structural Operations on DFAs
- 6 NFA: Nondeterministic Finite Automata
- 7 AFW: Alternating Finite Automata on Words

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 [3]).
- 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.)



AFW: Alternating Finite Automata on Words

Transitions as boolean functions

- For a given set of propositions (standing for states) 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.

AFW: Alternating Finite Automata on Words

NFA transitions

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 automaton's next state.

AFW transitions

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).

AFW: Alternating Finite Automata on Words

AFW: Alternating Finite Automata on Words

An AFW 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;
- $\rho : S \times \Sigma \rightarrow \mathcal{B}^+(S)$ is a **transition function**.

AFW: Alternating Finite Automata on Words

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

Run of AFWs

A **run** of an AFW $A = (\Sigma, S, s^0, \rho, F)$ on a finite word $w = a_0 a_1 \cdots a_n$ is a finite S -labeled tree r such that:

- $r(\epsilon) = s^0$
- for node x at depth (i.e., distance from the root) $i < n$, with $r(x) = s$ and $\rho(s, a_i) = \theta$ we have:
 - ▶ x has k children x_1, \dots, x_k , for some $k \leq |S|$,
 - ▶ $\{r(x_1), \dots, r(x_k)\}$ satisfies θ .

AFW: Alternating Finite Automata on Words

Example (Example of run)

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 .

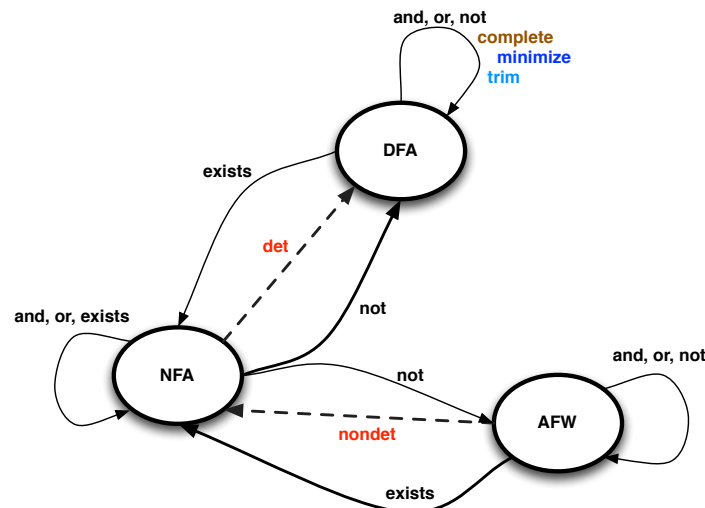
- Note that the depth of r (i.e., the maximal level of a node in r) is at most n , but not all branches need to reach such depth, since if $\rho(r(x), a_i) = \text{true}$, then x does not need to have any children.
- On the other hand, for node at depth $i < n$, we cannot have $\rho(r(x), a_i) = \text{false}$, since *false* is not satisfiable.

AFW acceptance condition

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.

AFW and NFA

- What is the relationship between alternating automata and nondeterministic automata?
- It turns out that AFW have the **same expressive power** as DFA and NFA
- AFW are **exponentially more succinct** than NFA, which in turn are **exponentially more succinct** than DFA
- In other words, any AFW can be translated into a **worst-case exponential NFA** (in number of states), and such NFA can be translated into a **worst-case exponential DFA** (i.e., the resulting DFA is **double exponential** in the original AFW)
- Such worst-case exponential blow-ups are **unavoidable**, in general.



Translating NFAs to AFWs

NFA to AFW

Given an NFA $A = (\Sigma, S, S^0, \rho, F)$ 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 (NFA to AFW [1, 2, 6])

Let A be NFA. Then the AFW A_A defined as above is such that $\mathcal{L}(A_A) = \mathcal{L}(A)$.

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.

AFW to NFA

Given AFW $A = (\Sigma, S, s^0, \rho, F)$, 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 (AFW to NFA [1, 2, 6])

Let A be an AFW. Then there the NFA A_N defined above is such that $\mathcal{L}(A_N) = \mathcal{L}(A)$.

The translation from AFWs to NFAs involves an **exponential blow-up**. This blow-up is **unavoidable** [1, 2, 6].



Boolean Closure

One advantage of alternating automata is that it is very **easy** to take **unions** and **intersection** and **complementation**.

AFW complementation

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:

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

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 $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$.

Theorem (AFW Complementation [1, 2, 6])

Let A be AFW. Then there exists an AFW \bar{A} such that $\mathcal{L}(\bar{A}) = \Sigma^* - \mathcal{L}(A)$.



Nonemptiness

To test for **nonemptiness** a AFW:

- Trasform it into a NFA (*exponential*)
- Test for nonemptiness on the resulting NFA, i.e., check reachability of its final states (*NLOGSPACE*).


Since the transformation can be done **on-the-fly**, this procedure is PSPACE, and in fact optimal, since the problem is PSPACE-hard.

Theorem (AFW Nonemptiness and AFW Nonuniversality [2])


The nonemptiness problem, as well as the nonuniversality problem, for AFWs is decidable in exponential time, and is PSPACE-complete.


How to check nonemptiness: transform into NFA and check nonemptiness

How to check nonuniversality: complement, transform into NFA and check nonemptiness


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


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

- 
- 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.

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

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

- 
- E. L. Leiss.
-
- Succinct representation of regular languages by boolean automata.
-
- Theor. Comput. Sci.*
- , 13:323–330, 1981.

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