CS112: Theory of Computation (LFA)

Lecture 4: Nondeterminism

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

March 18, 2025

Table of contents

Section 1

Previously on CS112

DFA

Definition

A finite automaton is 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q is a finite set called the states
2. Σ is a finite set called the alphabet
3. $\delta : Q \times \Sigma \to Q$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

Regular Language

Definition

A language is called a regular language if some finite automaton recognizes it.

Regular operations

Definition

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- Union: $A \cup B = \{x | x \in A \textit{ or } x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \textit{ and } y \in B\}$
- Star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \textit{ and each } x_i \in A\}$

Closure under union

Theorem

The class of regular languages is closed under the union operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Formal definition

Definition

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accepted states

We denote $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\mathcal{P}(Q)$ as the power set of Q.

Equivalence of NFAs and DFAs

Theorem

Every NFA has an equivalent DFA.

Equivalence of NFAs and DFAs

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA. We construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$ recognizing A. First we do our construction on an easy case when N has no ϵ arrows.

1. $Q' = \mathcal{P}(Q)$. Ever state of M is a set o states of N.
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \; for \; some \; r \in R\}$. If R is a state of M, it is also a set of states of N. When M reads symbol a in state R it shows where a takes each state on R. Because each state can go to a set o states we take the union of all this sets. More concise:
$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$
3. $q_0 = \{q_0\}$. M starts in the state corresponding to the collection containing just the start state of N
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$. The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Equivalence of NFAs and DFAs

Corollary

A language is regular if and only if some NFA recognizes it.

Section 2

Context setup

Context setup

- Now that we have defined NFA, we studied NFA and DFA equivalence we can use NFA to finish our proofs and understanding of **closure under regular operations**
- Next we will study DFA Minimization

Context setup

Corresponding to Sipser 1.2

DFA Minimization Link

Section 3

Closure under regular operations

Closure under regular operations

- Now we return to the closure of the class of regular languages under the regular operations that we began in previous lecture
- We abandoned the original attempt to do so when dealing with the concatenation operation was too complicated.
- Now using nondeterminism makes the proofs much easier.

Closure under union

- In a previous lecture we proved closure under union by simulating deterministically both machines simultaneously via a Cartesian product construction
- We now give a new proof to illustrate the technique of nondeterminism

Closure under union

Theorem

The class of regular languages is closed under the union operation

Proof idea:

- So, we have regular languages A_1 and A_2 and we want to prove that $A_1 \cup A_2$ is regular
- The idea is to take two NFAs, N_1 and N_2 for A_1 and A_2 and combine them into one NFA N
- Machine N must accept its input if either N_1 or N_2 accepts this input
- The new machine has a new start state that branches to the start states of the old machines with ϵ arrows.
- In this way, the new machine nondeterministically guesses which of the two machines accepts the input. If one of them accepts the input, N will accept it, too.

Closure under union

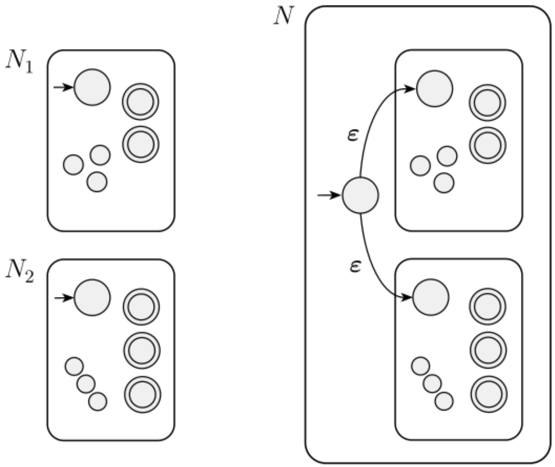


Figure: Assembly of NFA N

Closure under union I

Theorem

The class of regular languages is closed under the union operation

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$. The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0
2. The state q_0 is the start state of N
3. The set of accept states $F = F_1 \cup F_2$. The accept states of N are all the accept states of N_1 and N_2 . That way, N accepts if either N_1 accepts or N_2 accepts.

Closure under union II

Proof.

1. Define δ so that for any $q \in Q$ and and $a \in \Sigma_\epsilon$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

□

Closure under concatenation

Theorem

The class of regular languages is closed under the concatenation operation

Proof idea:

- We have regular languages A_1 and A_2 and want to prove that $A_1 \circ A_2$ is regular
- The idea is to take two NFAs, N_1 and N_2 for A_1 and A_2, and combine them into a new NFA N as we did for the case of union, but in a different way
- Assign N start state to be the start state of N_1. The accept states of N_1 have additional ϵ arrows that nondeterministically allow branching to N_2 whenever N_1 is in an accept state, signifying that it has found an initial piece of the input that constitutes a string in A_1. s
- The accept states of N are the accept states of N_2

Closure under concatenation

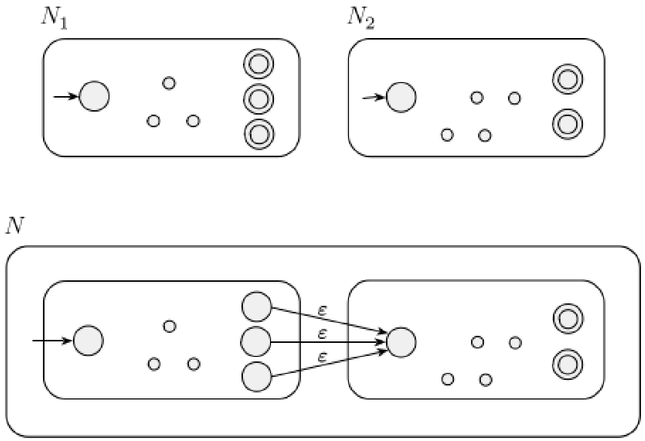


Figure: Assembly of NFA N

Closure under concatenation I

Theorem

The class of regular languages is closed under the concatenation operation

Closure under concatenation II

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2 Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$:

1. $Q = Q_1 \cup Q_2$. The states of N are all the states of N_1 and N_2
2. The state q_1 is the start state of N, same as N_1
3. The set of accept states F_2, same as N_2
4. Define δ so that for any $q \in Q$ and and $a \in \Sigma_\epsilon$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \textit{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \textit{ and } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \textit{ and } a = \epsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

Closure under star

Theorem

The class of regular languages is closed under the star operation

Proof idea:

- We have a regular language A_1 and want to prove that A_1^* is regular
- We take an NFA N_1 for A_1 and modify it to recognize A_1^*. The resulting NFA N will accept its input whenever it can be broken into several pieces and N_1 accepts each piece
- We can construct N like N_1 with additional ϵ arrows returning to the start state from the accept states
- This way, when processing gets to the end of a piece that N_1 accepts, the machine N has the option of jumping back to the start state to try to read another piece that N_1 accepts

Closure under star

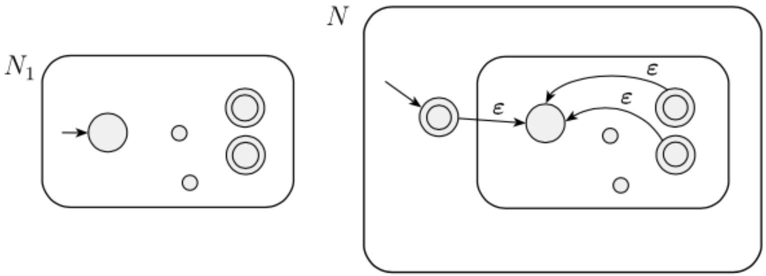


Figure: Assembly of NFA N

Closure under star I

Theorem

The class of regular languages is closed under the star operation

Closure under star II

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*:

1. $Q = \{q_0\} \cup Q_1$. The states of N are the states of N_1 plus a new start state
2. The q_0 is the new start state
3. $F = \{q_0\} \cup F_1$. The accept states are the old accept states plus the new start state
4. Define δ so that for any $q \in Q$ and and $a \in \Sigma_\epsilon$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \epsilon \\ \{q_1\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \neq \epsilon \end{cases}$$

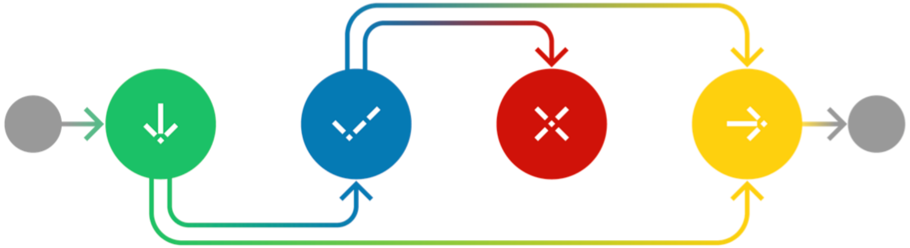
$\square$

Closure under star

One (slightly bad) idea is simply to add the start state to the set of accept states. This approach certainly adds ϵ to the recognized language, but it may also add other, undesired strings. Why? ($\Leftarrow$ first to find an explanation will get a CS112 T-shirt)

T-Shirt

CS112 Logo



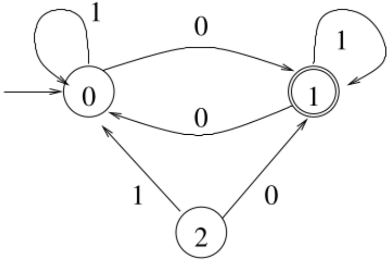
Section 4

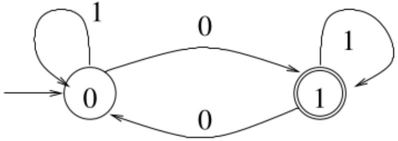
Minimization of DFAs

Context setup

- DFA minimisation is an important topic because it can be applied both theoretically and practically (e.g., compilers)
- Minimising a DFA increases its efficiency by reducing its amount of states and it also enables us to determine if two DFAs are equivalent.

Example

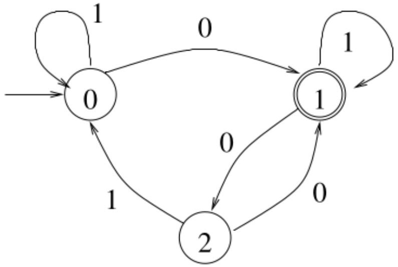


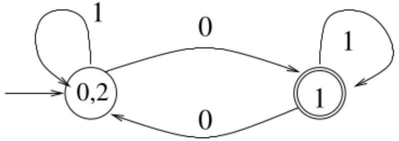


You can see that it is not possible to ever visit state 2. States like this are called unreachable. We can simply remove them from the automaton without changing its behavior. (This will be, indeed, the first step in our minimization algorithm.)

Example

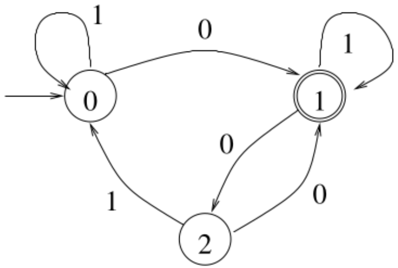
As it turns out, however, removing unreachable states is not sufficient. The below example is a bit more interesting

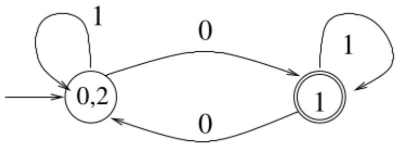




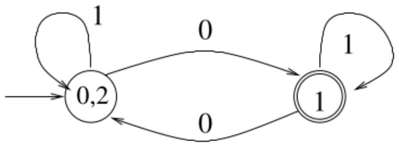
Example

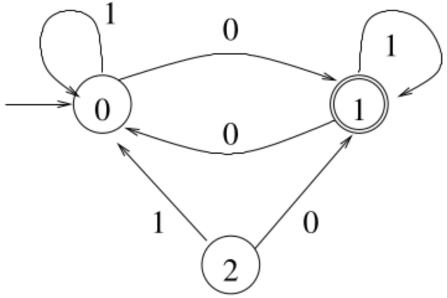
Let's compare what happens if we start processing some string w from state 0 or from state 2. Is the result the same?





Example





- If $w = \epsilon$, we will stay in either of the two states and ϵ will not be accepted
- If w starts with 0, in both cases we will go to state 1 and both computations will be now identical
- If w starts with 1, in both cases we will go to state 0 and both computations will be now identical
- No matter what w is, either we accept w in both cases or we reject w in both cases
- So it is ok to combine states 0 and 2 into one state will be identical

Formal definition

Definition

Let A be a DFA. We say that $w \in \Sigma^*$ **distinguishes** between two states $q_1, q_2 \in Q$ if either $\delta(q_1, w) \in F$ and $\delta(q_2, w) \notin F$ or $\delta(q_1, w) \notin F$ and $\delta(q_2, w) \in F$

Definition

Two states $q_1, q_2 \in Q$ are called **distinguishable** iff there is a word that distinguishes between them. States that are indistinguishable will also be sometimes called **equivalent**

Minimization Algorithm

The reasoning above leads to the following method:

- Start with an DFA A without unreachable states
- If A has undistinguishable states q_1, q_2, combine them into one state. (For instance, remove q_1 and reroute all transitions into q_1 to go into q_2 instead)
- Repeat this process until no more distinguishable states can be found. At this point we will not be able to reduce A further
- But does it necessarily mean that A is minimum?
- We will prove it, however let us first see the algorithm

Minimization Algorithm

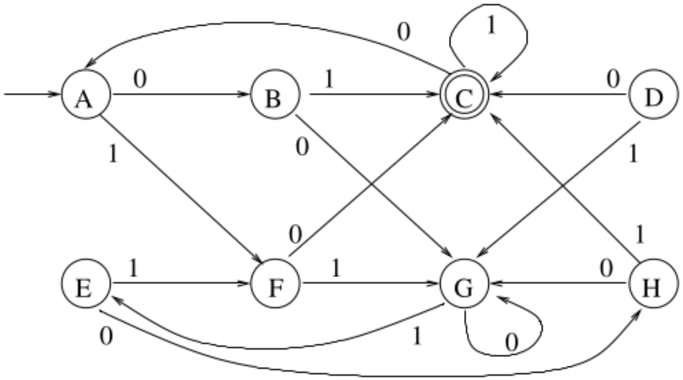
1. Remove unreachable states
2. Mark the distinguishable pairs of states
    - To achieve this task, we first mark all pairs p, q, where $p \in F$ and $q \notin F$ as distinguishable.
    - Then, we proceed as follows:

```
repeat
    for all non-marked pairs [p,q] do
        for each letter a do
            if the pair [d(p,a), d(q,a))] is marked
                then mark [p,q]
until no new pairs are marked
```

Minimization Algorithm

3. Construct the reduced automaton A'
    - We first determine the equivalence classes of the indistinguishability relation. For each state q, the equivalence class of q consists of all states p for which the pair p, q is not marked in Step 2
    - The states of A' are the equivalence classes. The initial state q_0' is this equivalence class that contains q_0 . The final states F' are these equivalence classes that consist of final states of A
    - The transition function δ' is defined as follows. To determine $\delta'(X, a)$ for some equivalence class X, pick any $q \in X$ and set $\delta'(X, a) = Y$ where Y is the equivalence class that contains $\delta(q, a)$

Example



Step 1: We have one unreachable state, state *D*. We remove this state and proceed to Step 2

Example

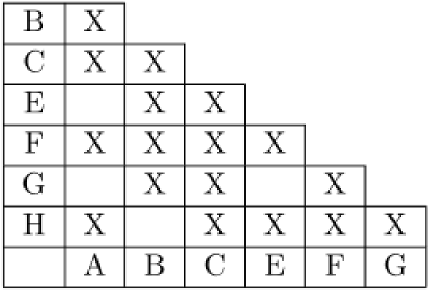
Step 2: We first mark pairs of final and non-final states

B						
C	X	X				
E			X			
F			X			
G			X			
H			X			
	A	B	C	E	F	G

Next we examine all unmarked pairs. For example, for pair A, B we get $\delta(A, 1) = F$ and $\delta(B, 1) = C$. Since C, F is marked, we mark A, B too. We proceed for all the pairs

Example

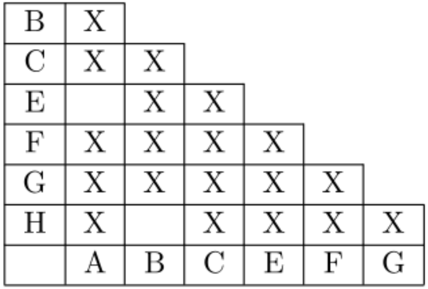
Step 2: We first mark pairs of final and non-final states



B	X					
C	X	X				
E		X	X			
F	X	X	X	X		
G		X	X		X	
H	X		X	X	X	X
	A	B	C	E	F	G

Next we examine all unmarked pairs. For example, for pair A, B we get $\delta(A, 0) = B$ and $\delta(G, 0) = G$. Since B, G is marked, we mark A, G too. We proceed for all the pairs

Example

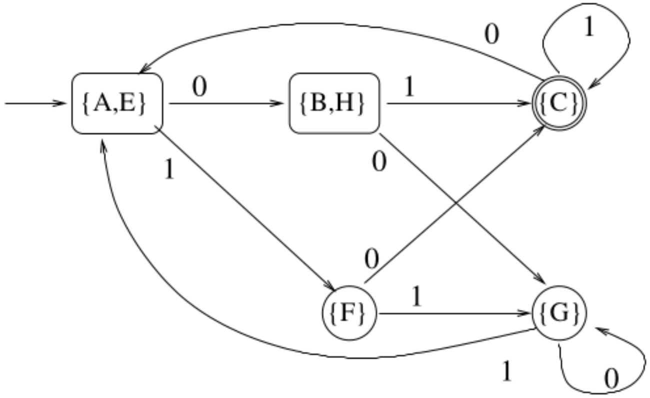


B	X					
C	X	X				
E		X	X			
F	X	X	X	X		
G	X	X	X	X	X	
H	X		X	X	X	X
	A	B	C	E	F	G

Step 2: but no new distinguishable pairs will be discovered. So we are done with Step 2 now.

Example

We group the states into the equivalence classes. Since A, E are equivalent and B, H are equivalent, the classes are: $\{A, E\}, \{B, H\}, \{C\}, \{F\}, \{G\}$. The minimal automaton A' is:



Correctness

To justify correctness, we need to prove a couple of things First, we need to show that we compute the equivalence classes correctly and that A' is a well-defined DFA. Then, we need to show that it accepts the same language as A. Finally, we need to show that there is no DFA A'' equivalent to A with fewer states

Formal definition

Lemma

Suppose that A is a DFA without unreachable states. Then A is minimum if and only if all pairs of states are distinguishable.

Proof.

Homework :) □

Formal definition

Lemma

State indistinguishability is an equivalence relation

Proof.

Homework :) $\square$

Formal definition

Lemma

Let $\delta(p, a) = p'$ and $\delta(q, a) = q'$. Then, if p', q' are distinguishable then so are p, q.

Proof.

Homework :) $\square$

Formal definition

- Write all proof in Latex and send them over
- First 5 submissions will get a CS112 T-Shirt and 10pct :)
- Latex file must compile and proofs must be complete
- Link: Minimize DFA

Formal definition I

Theorem

Our minimization algorithm is correct that is $L(A') = L(A)$ and A' is minimum

Formal definition II

Proof.

We prove all the required conditions, one by one.

1. **Why are the equivalence classes computed correctly?** Here, we need to show that the pair p, q is marked iff p, q are distinguishable. The proof of this is quite easy, by induction on the length of the shortest string that distinguishes p, q, using Lemma 3.
2. **Why is A' well defined?** That follows from Lemma 2. The fact that indistinguishability is an equivalence relation implies that the states of A' are well-defined. We also have that in each equivalence class either all states are final or all states are non-final. The second condition implies that the transitions are well-defined.
3. **Why $L(A') = L(A)$** To prove this, we show that for each w we have $\delta(q_0, w) \in \delta'(q'_0, w)$. This can be proved by induction on the length of w.
4. **Why is A' minimal**. We use Lemma 1. We need to show that in A' all states are distinguishable. This is quite obvious from the construction.

$\square$