Tema #1 - Limbaje Formale și Automate

Vrînceanu Radu-Tudor radu-tudor.vrinceanu@unibuc.ro

27 februarie 2025

1 Administrativ

Tema 1 de laborator pentru materia Limbaje Formale și Automate are ca și timp de finalizare:

- **23 martie 2025, ora 23:59** pentru grupa 151

Orice submisie după această limită va fi depunctată cu 0.5 puncte per zi, ce este după laboratorul 3 care va avea loc pentru grupa 151 pe **25 martie 2025**..

1.1 Ce aveți de livrat pentru această temă?

Aveți de livrat două fișiere de tip cod-sursă (într-unul din limbajele C/C++, Java, Python 3.x, Go, Typescript / Javascript (în Node.js / Deno)) care reprezintă soluția pentru temele voastre. În fiecare fișier la începutul acestuia trebuie să aveți:

- Numele și prenumele vostru;
- Grupa din care faceți parte;
- Tipul compilatorului și / sau versiunea de limbaj (sau alte informații despre ce ați folosit pentru a face funcțional proiectul)

1.2 Metoda de evaluare

$$Nota_{tema_1} = \frac{Punctaj_{etapa_1} + Punctaj_{etapa_2}}{2}$$

1.3 Avem voie cu cod generat, respectiv cu cod copiat?

Aveți voie să mă întrebați, să consultați cursul, să citiți cărțile din bibliografie (vezi Hopcroft & Ullman, M. Sipser), aveți voie să preluați bucăți de cod (așa numitele snippet-uri), aveți voie să luați cod de pe StackOverflow, medium.com, etc.

Ce înseamnă cod copiat:

- cod preluat de la unul dintre colegii voștrii din serie;
- cod preluat de la colegii din generațiile anterioare (repository-uri de GitHub, GitLab, BitBucket, etc.);
- cod generat prin modele lingvistice sau de cod (CodeLLama, Qwen, Gemini, Gemma, Claude, Bolt, Perplexity, DeepSeek, ChatGPT și alte tipuri de astfel de transformer decoders :D)

Dacă aveți IntelliSense în IDE-ul vostru puteți să îl folosiți pentru auto-complete, dar NU auto-suggestion, recte Copilot.

Prefer să aveți mai mult timp să vă faceți tema decât să copiați cod și să NU vă asumați aceste concepte.

1.4 Fișierul de configurare

Are 3 secțiuni care pot fi aranjate în orice ordine. Comentariile vor fi definite prin simbolul #. Dacă întâlniți comentariul # în dreptul unei linii atunci linia respectivă trebuie ignorată (i.e. dacă este goală nu se întâmplă nimic, dacă este într-o secțiune nu se preia acea informație). **Se garantează faptul că rândurile ce conțin cuvintele-cheie States, Transitions, Sigma și End NU vor acea niciodată comentate.** Fiecare stare poate fi însoțită după numele ei de "S", "F", ambele sau nimic. "S" poate fi plasat doar pentru o singură stare!

2 Cerințe

2.1 Etapa 1 - DFA (Deterministic Finite Automata)

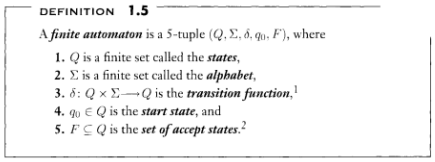
- (10 fișiere de configuații a câte 0.5 puncte fiecare, $10 * 0.5 = 5p$) Fiind date niște fișiere de configurare de forma regăsită în Appendix scrieți un script sau o librărie prin care validați dacă automatul descris este un DFA.
- (5 fișiere de configurații VALIDE cu 4 stringuri per fiecare fișier de configurație, rezultând 20 de teste a câte 0.25 puncte fiecare, $20 * 0.25 = 5p$) Fiind date niște fișiere de configurare de forma regăsită în Appendix scrieți un script sau o librărie prin care testați dacă DFA-ul descris în acel fișier de configurare acceptă un string s sau nu.

2.2 Etapa 2 - NFA (Nondeterministic Finite Automata)

- (10 fișiere de configuații a câte 0.5 puncte fiecare, $10 * 0.5 = 5p$) Fiind date niște fișiere de configurare de forma regăsită în Appendix scrieți un script sau o librărie prin care validați dacă automatul descris este un NFA.
- (5 fișiere de configurații VALIDE cu 4 stringuri per fiecare fișier de configurație, rezultând 20 de teste a câte 0.25 puncte fiecare, $20 * 0.25 = 5p$) Fiind date niște fișiere de configurare de forma regăsită în Appendix scrieți un script sau o librărie prin care testați dacă NFA-ul descris în acel fișier de configurare acceptă un string s sau nu.

3 Appendix

```
# This is an example of the configuration file and a comment
Sigma:
a
b
c
End
#
#
States:
q0, S
q1
q2
q3
q4, F
q5
End
# another comment
#
Transitions:
q0, a, q1
q0, b, q2
q0, c, q3
q1, b, q2
q2, a, q3
q1, c, q4
q2, b, q4
q3, a, q4
q4, a, q5
q5, a, q4
End
```

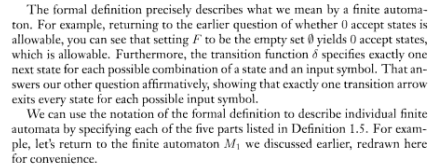


DEFINITION 1.5

A ***finite automaton*** is a 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. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

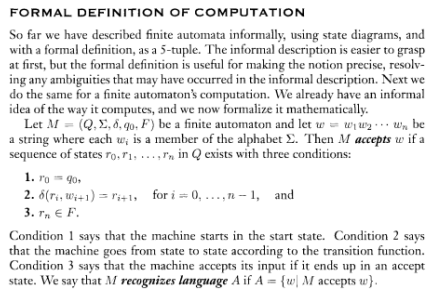
Figure 1: Definiția unui automat finit (determinist)



The formal definition precisely describes what we mean by a finite automaton. For example, returning to the earlier question of whether 0 accept states is allowable, you can see that setting F to be the empty set $\emptyset$ yields 0 accept states, which is allowable. Furthermore, the transition function δ specifies exactly one next state for each possible combination of a state and an input symbol. That answers our other question affirmatively, showing that exactly one transition arrow exits every state for each possible input symbol.

We can use the notation of the formal definition to describe individual finite automata by specifying each of the five parts listed in Definition 1.5. For example, let's return to the finite automaton M_1 we discussed earlier, redrawn here for convenience.

Figure 2: Informații despre mulțimea stărilor finale ale unui automat finit (determinist)



FORMAL DEFINITION OF COMPUTATION

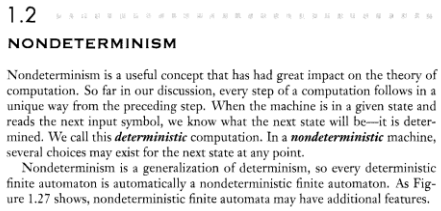
So far we have described finite automata informally, using state diagrams, and with a formal definition, as a 5-tuple. The informal description is easier to grasp at first, but the formal definition is useful for making the notion precise, resolving any ambiguities that may have occurred in the informal description. Next we do the same for a finite automaton's computation. We already have an informal idea of the way it computes, and we now formalize it mathematically.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w | M \text{ accepts } w\}$.

Figure 3: Ce înseamnă că acceptăm sau să respingem un string



1.2

NONDETERMINISM

Nondeterminism is a useful concept that has had great impact on the theory of computation. So far in our discussion, every step of a computation follows in a unique way from the preceding step. When the machine is in a given state and reads the next input symbol, we know what the next state will be—it is determined. We call this ***deterministic*** computation. In a ***nondeterministic*** machine, several choices may exist for the next state at any point.

Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton. As Figure 1.27 shows, nondeterministic finite automata may have additional features.

Figure 4: Ce înseamnă nondeterminismul

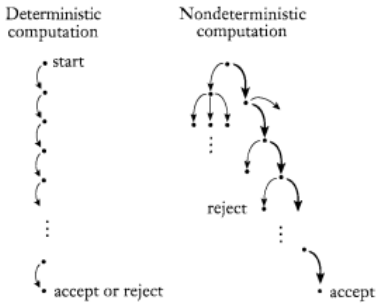


Figure 5: Computare determnistă vs. nondeterministă

DEFINITION 1.37

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. $\delta\colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Figure 6: Definiția unui automati finit nedeterminist (lambda / epsilon), pentru această temă nu luăm în considerare λ-NFA-urile sau ϵ-NFA-urile (denumite de M. Sipser)

Let A be the language consisting of all strings over {0,1} containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA N_2 recognizes A.

FIGURE 1.31
The NFA N_2 recognizing A

One good way to view the computation of this NFA is to say that it stays in the start state q_1 until it "guesses" that it is three places from the end. At that point, if the input symbol is a 1, it branches to state q_2 and uses q_3 and q_4 to "check" on whether its guess was correct.

As mentioned, every NFA can be converted into an equivalent DFA, but sometimes that DFA may have many more states. The smallest DFA for A contains eight states. Furthermore, understanding the functioning of the NFA is much easier, as you may see by examining the following figure for the DFA.

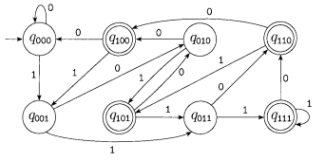


FIGURE 1.32
A DFA recognizing A

Figure 7: Echivalența dintre un NFA și DFA