Introduction to Theoretical Computer Science

Lecture 1: Finite Automata

Dr. Liam O'Connor

University of Edinburgh
Semester 1, 2023/2024

Course Aims

- understanding of computability, complexity and intractability;
- knowledge of lambda calculus, types, and type safety.

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular
- Explain decidability, undecidability and the halting problem.

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular
- Explain decidability, undecidability and the halting problem.
- Demonstrate the use of reductions for undecidability proofs.

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular
- Explain decidability, undecidability and the halting problem.
- Demonstrate the use of reductions for undecidability proofs.
- Explain the notions of P, NP, NP-complete.

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular
- Explain decidability, undecidability and the halting problem.
- Demonstrate the use of reductions for undecidability proofs.
- Explain the notions of P, NP, NP-complete.
- Use reductions to show problems to be NP-hard.

Course Outcomes

By the end of the course you should be able to

- Explain (non-)deterministic finite and pushdown automata and use the pumping lemma to show languages non-regular
- Explain decidability, undecidability and the halting problem.
- Demonstrate the use of reductions for undecidability proofs.
- Explain the notions of P, NP, NP-complete.
- Use reductions to show problems to be NP-hard.
- Write short programs in lambda-calculus.

Course Outline

- Introduction. Finite automata.
- Regular languages and expressions.
- Context-free languages and pushdown automata.
- Register machines and their programming.
- Universal machines and the halting problem.
- Decision problems and reductions.
- Undecidability and semi-decidability.
- Complexity of algorithms and problems.
- The class P
- Non-determinism and NP
- NP-completeness
- Beyond NP.
- Lambda-calculus.
- Recursion.
- Types.

Assessment

The course is assessed by a written examination (80%) and two coursework exercises, the first formative and the second summative (for the remaining 20%).
Coursework deadlines: End of weeks 5 and 9.

Textbooks

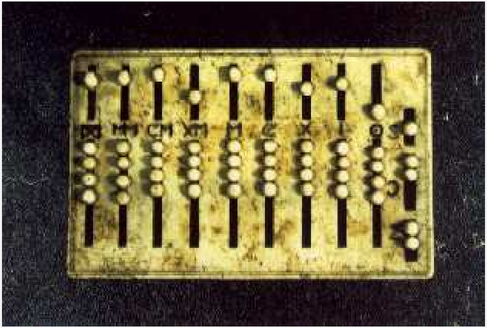
It will be useful, but not absolutely necessary, to have access to:

- Michael Sipser *Introduction to the Theory of Computation*, PWS Publishing (International Thomson Publishing)
- Benjamin C. Pierce *Types and Programming Languages*, MIT Press

There is also much information on the Web, and in particular Wikipedia articles are generally fairly good in this area. Generally I will refer to textbooks for the detail of material I discuss on slides.

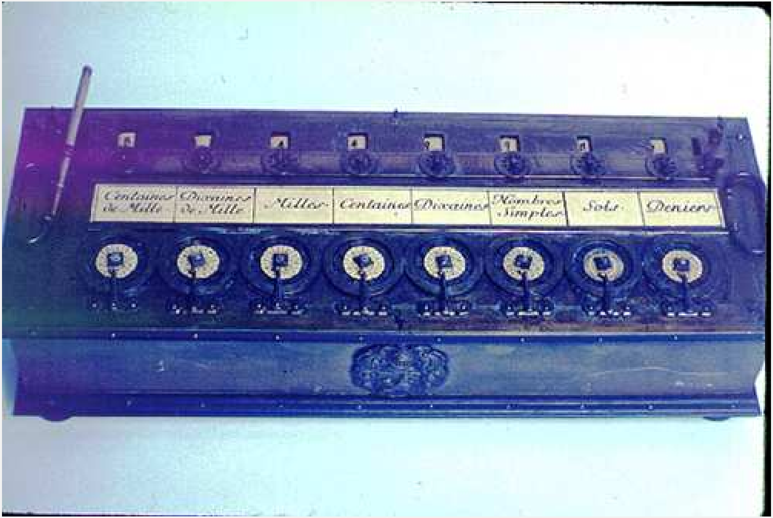
What is computation? What are computers?

Some computing devices:
The abacus – some millennia BP.



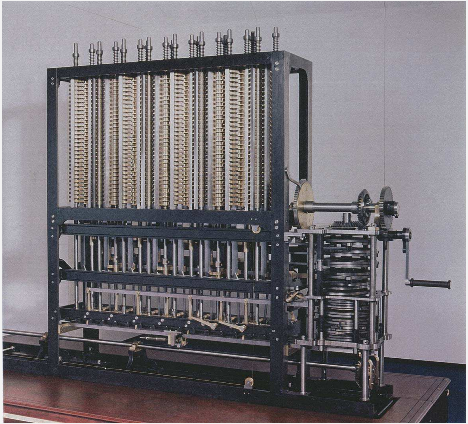
[Association pour le musée international du calcul de l'informatique et de l'automatique de Valbonne Sophia Antipolis (AMISA)]

First mechanical digital calculator – 1642 Pascal



[original source unknown]

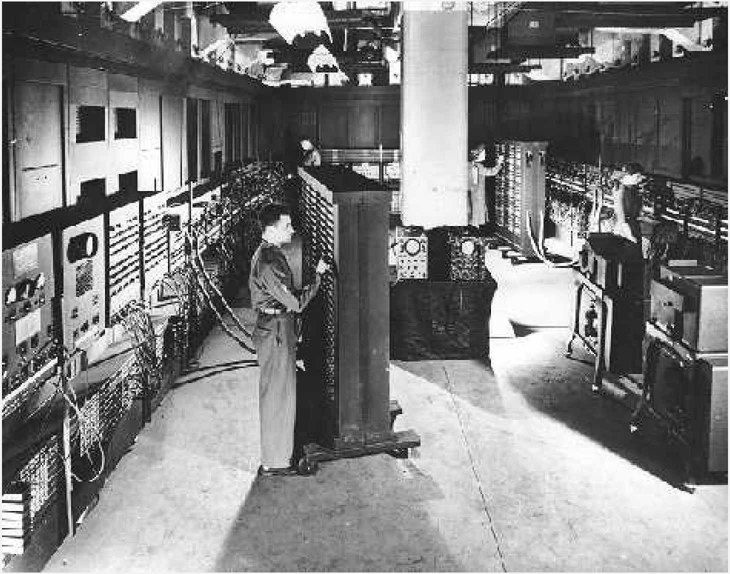
The Difference Engine, [The Analytical Engine] – 1812, 1832 Babbage / Lovelace.



[Science Museum ??]

Analytical Engine (never built) anticipated many modern aspects of computers. See `http://www.fourmilab.ch/babbage/`.

ENIAC – 1945, Eckert & Mauchley



[University of Pennsylvania]

What do computers manipulate?

Symbols? Numbers? Bits? Does it matter?

What do computers manipulate?

Symbols? Numbers? Bits? Does it matter?
What about real numbers? Physical quantities? Proofs?
Emotions?

What do computers manipulate?

Symbols? Numbers? Bits? Does it matter?
What about real numbers? Physical quantities? Proofs? Emotions?
Do we buy that numbers are enough? If we buy that, are bits enough?

What do computers manipulate?

Symbols? Numbers? Bits? Does it matter?
What about real numbers? Physical quantities? Proofs? Emotions?
Do we buy that numbers are enough? If we buy that, are bits enough?
How much memory do we need?

What can we compute?

If we can cast a problem in terms that our computers manipulate, can we solve it?

What can we compute?

If we can cast a problem in terms that our computers manipulate, can we solve it? Always? Sometimes?

What can we compute?

If we can cast a problem in terms that our computers manipulate, can we solve it? Always? Sometimes? With how much time?

What can we compute?

If we can cast a problem in terms that our computers manipulate, can we solve it? Always? Sometimes? With how much time? With how much memory?

What can we compute?

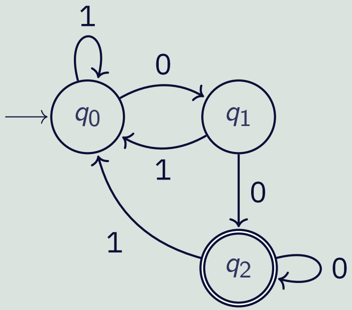
If we can cast a problem in terms that our computers manipulate, can we solve it? Always? Sometimes? With how much time? With how much memory?

In this course

We will seek mathematical answers to these questions. For that, we will need a model of computation.

Finite Automata

Example



A finite automaton takes a string as input and says "yes" or "no".

Define the *language* of a finite automaton A, written, $\mathcal{L}(A)$ to be the set of strings for which A says "yes".

A string is a (possibly-empty) sequence of *symbols* from a set called an *alphabet*, usually written Σ.

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

DFAs, formally

Definition

A *deterministic finite automaton* (DFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

Exercise: What is the formal definition of our example?

Languages

Definition

A DFA *accepts* a string $w \in \Sigma^*$ iff $\delta^*(q_0, w) \in F$, where δ^* is δ applied successively for each symbol in w. The language of a DFA $\mathcal{L}(A) \subseteq \Sigma^*$ is the set of all strings accepted by A.

Languages

Definition

A DFA *accepts* a string $w \in \Sigma^*$ iff $\delta^*(q_0, w) \in F$, where δ^* is δ applied successively for each symbol in w. The language of a DFA $\mathcal{L}(A) \subseteq \Sigma^*$ is the set of all strings accepted by A.

Exercise: What is the language of our example?

Determinism

In a DFA, the transition function is a total function which gives exactly one next state for each input symbol (it's *deterministic*).

Questions

Does relaxing any of these requirements affect the set of languages we can recognise? How would we prove this?

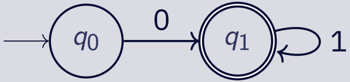
Determinism

In a DFA, the transition function is a total function which gives exactly one next state for each input symbol (it's *deterministic*).

Questions

Does relaxing any of these requirements affect the set of languages we can recognise? How would we prove this?

- What if we made δ *partial*?



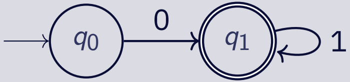
Determinism

In a DFA, the transition function is a total function which gives exactly one next state for each input symbol (it's *deterministic*).

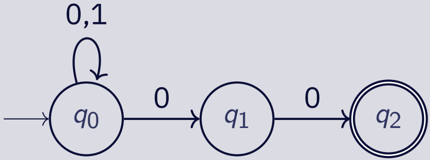
Questions

Does relaxing any of these requirements affect the set of languages we can recognise? How would we prove this?

- What if we made δ *partial*?



- What if we made δ *non-deterministic*?



Nondeterministic Finite Automata

Definition

A *nondeterministic finite automaton* (NFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

Nondeterministic Finite Automata

Definition

A *nondeterministic finite automaton* (NFA) is a quintuple $(Q, \Sigma, q_0, \delta, F)$ where:

- Q is a **finite** set of *states*,
- Σ is the *alphabet*, the set of *symbols*,
- $q_0 \in Q$ is the *initial state*
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

Note the only difference here is the transition function, which gives a set of next states for a given symbol.

Nondeterministic Finite Automata

Definition

A *run* of an NFA A on a string $w = a_1 a_2 \dots a_k$ is a sequence of states $q_0 q_1 \dots q_k$ in Q such that:

- q_0 is the initial state
- for all $i = 1 \dots k$ we have $q_i \in \delta(q_{i-1}, a_i)$.

A run is *accepting* if the last state $q_k \in F$.

Nondeterministic Finite Automata

Definition

A *run* of an NFA A on a string $w = a_1 a_2 \ldots a_k$ is a sequence of states $q_0 q_1 \ldots q_k$ in Q such that:

- q_0 is the initial state
- for all $i = 1 \ldots k$ we have $q_i \in \delta(q_{i-1}, a_i)$.

A run is *accepting* if the last state $q_k \in F$.

The nondeterminism means that we have multiple alternative computations. For our purposes we will use *angelic* non-determinism, which says that we achieve success if any of our alternatives succeed.

$$\mathcal{L}(A) = \{w \mid \text{there exists an accepting run of } A \text{ on } w\}$$

Nondeterministic Finite Automata

Definition

A *run* of an NFA A on a string $w = a_1 a_2 \ldots a_k$ is a sequence of states $q_0 q_1 \ldots q_k$ in Q such that:

- q_0 is the initial state
- for all $i = 1 \ldots k$ we have $q_i \in \delta(q_{i-1}, a_i)$.

A run is *accepting* if the last state $q_k \in F$.

The nondeterminism means that we have multiple alternative computations. For our purposes we will use *angelic* non-determinism, which says that we achieve success if any of our alternatives succeed.

$$\mathcal{L}(A) = \{w \mid \text{there exists an accepting run of } A \text{ on } w\}$$

Exercise: Is `10100` in the language of our previous example?

NFA = DFA

Claim

Making finite automata non-deterministic does not change their expressivity. That is, for every non-deterministic automaton A there is a deterministic automaton D such that $\mathcal{L}(D) = \mathcal{L}(A)$ and vice versa.

NFA = DFA

Claim

Making finite automata non-deterministic does not change their expressivity. That is, for every non-deterministic automaton A there is a deterministic automaton D such that $\mathcal{L}(D) = \mathcal{L}(A)$ and vice versa.

- **DFA $\Rightarrow$ NFA**:

NFA = DFA

Claim

Making finite automata non-deterministic does not change their expressivity. That is, for every non-deterministic automaton A there is a deterministic automaton D such that $\mathcal{L}(D) = \mathcal{L}(A)$ and vice versa.

- **DFA** $\Rightarrow$ **NFA**:Easy, the DFA is already an NFA where the transition function always returns a singleton set.
- **NFA** $\Rightarrow$ **DFA**:

NFA = DFA

Claim

Making finite automata non-deterministic does not change their expressivity. That is, for every non-deterministic automaton A there is a deterministic automaton D such that $\mathcal{L}(D) = \mathcal{L}(A)$ and vice versa.

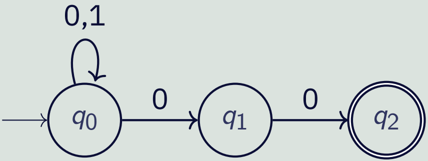
- **DFA** $\Rightarrow$ **NFA**:Easy, the DFA is already an NFA where the transition function always returns a singleton set.
- **NFA** $\Rightarrow$ **DFA**:We will use the *subset construction*.

Subset Construction

Key Idea

For an NFA A, the corresponding DFA D tracks the set of states that A could possibly be in, given the string read so far. So, each state of D is a set of states from A.

Example (From earlier)



Formally

The Subset Construction

Given an NFA $(Q_A, \Sigma, q_0, \delta_A, F_A)$, construct a DFA $(\mathcal{P}(Q_A), \Sigma, \{q_0\}, \delta_D, F_D)$ where:

$$\delta_D(S, a) = \bigcup_{q \in S} \delta_A(q, a) \quad \text{for each } S \subseteq Q$$

and

$$F_D = \{S \subseteq Q \mid S \cap F_A \neq \varnothing\}$$

Formally

The Subset Construction

Given an NFA $(Q_A, \Sigma, q_0, \delta_A, F_A)$, construct a DFA $(\mathcal{P}(Q_A), \Sigma, \{q_0\}, \delta_D, F_D)$ where:

$$\delta_D(S, a) = \bigcup_{q \in S} \delta_A(q, a) \quad \text{for each } S \subseteq Q$$

and

$$F_D = \{S \subseteq Q \mid S \cap F_A \neq \varnothing\}$$

Question: If our NFA has n states, how many states could our DFA have?

Proof?

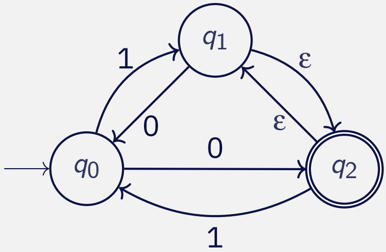
Proving that this is correct (i.e. that the DFA D obtained from an NFA A recognises the same language as A) relies on a proof by induction on the length of the input string w, that $\delta_D^*(\{q_0\}, w)$ is the set of all states q such that there exists a run of A on w from q_0 to q.
We will cover this if we have extra time.

ε-NFAs

Another Generalisation

What if we allow non-deterministic state changes that do not consume any input symbols?

We label these *silent moves* with ε (the empty string):

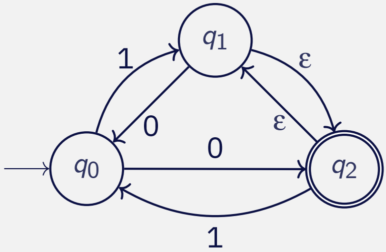


ε-NFAs

Another Generalisation

What if we allow non-deterministic state changes that do not consume any input symbols?

We label these *silent moves* with ε (the empty string):



Exercise: Is 001 accepted above? Can we express this as a DFA?

ε-NFA to DFA

The subset construction also applies to ε-NFAs.

ε-NFA to DFA

The subset construction also applies to ε-NFAs.

Definition

Define the *ε-closure* $E(q)$ of a state q as the set of all states reachable from q by silent moves.

ε-NFA to DFA

The subset construction also applies to ε-NFAs.

Definition

Define the *ε-closure* $E(q)$ of a state q as the set of all states reachable from q by silent moves.That is, $E(q)$ is the least set satisfying:

- $q \in E(q)$
- For any $s \in E(q)$, we also have $\delta(s, \varepsilon) \subseteq E(q)$.

We also extend this to sets, where $E(S) = \bigcup_{q \in S} E(q)$.

ε-NFA to DFA

The subset construction also applies to ε-NFAs.

Definition

Define the *ε-closure* $E(q)$ of a state q as the set of all states reachable from q by silent moves.That is, $E(q)$ is the least set satisfying:

- $q \in E(q)$
- For any $s \in E(q)$, we also have $\delta(s, \varepsilon) \subseteq E(q)$.

We also extend this to sets, where $E(S) = \bigcup_{q \in S} E(q)$.

In our subset construction, everything is the same except that each subset (each state of our DFA) is ε-closed:

$$\delta_D(S, a) = E\left(\bigcup_{s \in S} \delta_A(s, a)\right)$$

Summary

DFAs, NFAs and ε-NFAs all recognise the same class of languages, called the *regular languages*. They are equal in expressive power, although some representations (NFAs) are more compact than others (DFAs).

[1] Or this time, if we have time.

Summary

DFAs, NFAs and ε-NFAs all recognise the same class of languages, called the *regular languages*. They are equal in expressive power, although some representations (NFAs) are more compact than others (DFAs).

Questions for next time[1]

- Are the regular languages closed under union? sequential composition? intersection? complement?
  (How would we prove this?)
- What languages are not regular?
  (How would we prove this?)

[1] Or this time, if we have time.