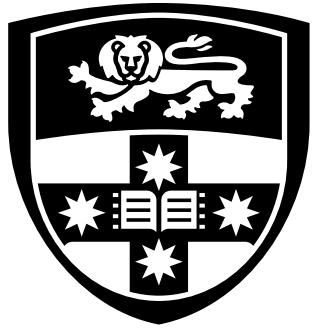
COMP2022: Formal Languages and Logic

2017, Semester 1, Week 1

Joseph Godbehere

Adapted from slides by A/Prof Kalina Yacef

March 7, 2017



THE UNIVERSITY OF
SYDNEY

Outline

- **Why study formal languages and logic?**
- Mathematical notions and definitions
- Finite State Automata / Regular Languages

Why should we study theory?

Computers are complex machines and theory provides a new viewpoint, an elegant side to computation

Studying theory expands your mind. IT evolves quickly. Whilst specific technical knowledge is quickly outdated, theory is not.

- Know how to express yourself clearly
- Know how to prove your work
- Know when you have, or have not, solved a problem.

Theory provides conceptual tools which are used in computer engineering

How this course will help

Most problems in computer science involve answering:

- Can the problem be solved by a computer program?
- If not, can you modify the problem so that it can?
- If so, can a program solve the problem in workable time?
- Can you prove that your program is correct?
- Can you prove that your program is efficient?

How this course will help

Formal Languages: *Structure*

- Regular languages / finite automata:
  - Text processing / pattern matching
  - Compilers and programming languages (lexical structure)
  - Agent based game 'AI'
  - Hardware design
- Context-free grammars:
  - Compilers and programming langauges (syntax)
  - Natural Language Processing / AI
- Turing machines: our computers
- Introduction to decidability/computability and intractability/complexity

... and many more

How this course will help

Logic: *Meaning*

- Program logic - logic gates, conditional statements
- Software verification
- Databases
- Artificial Intelligence
- Automated Reasoning

... and many more

Course topics

- Regular Languages
  - Finite automata
  - Regular grammars, regular expressions
- Context-free Languages
  - Pushdown Automata, C-F grammars, parsers
- Turing Machines
  - Church-Turing thesis
  - Computability, decidability, tractability
- Logic
  - Propositional and predicate logic
  - Logic formal proofs

Examples of Practical Uses

- Regular expressions widely used for pattern matching
  - e.g. Unix *ls *.java*
  - e.g. tag descriptions in DTD *person (name, address, child*)*
- Finite automata are used for model checking to verify correctness of communication protocols and electronic circuits

Examples of Practical Uses (2)

- Context-free grammars describe the syntax of
  - programming languages
  - natural languages (machine translation, Natural Language Processing tasks)

- XML DTDs overall syntax

Examples of Practical Uses (3)

- Logic: write correct boolean expressions
  - e.g. If conditions
- prove that a program will deliver correct output, is error free
  - Formal verification
  - Model checking
- Automated theorem provers
- Constraint programming solvers

A fundamental question in CS

- What are the fundamental capabilities and limitations of computers?

- We will study abstract computing machines with increasing levels of capabilities, corresponding to languages of increasing levels of expressiveness

Understanding limitations of computing

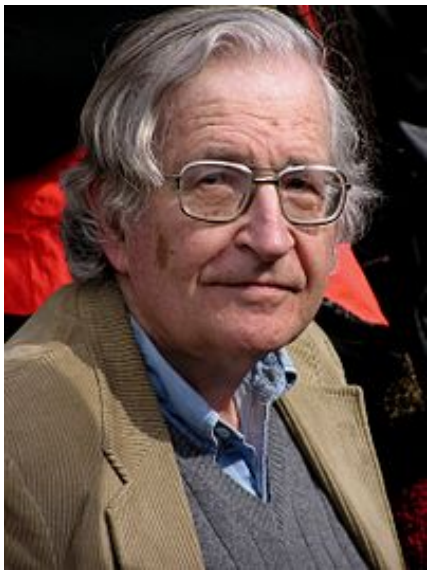
- When developing solutions to real problems, we need to understand the limitations of what software can do, and design solutions which are realistic and efficient.
  - *Undecidable* problems: no program/algorithm can do it (for all possible inputs)
  - *Intractable* problems: there could be programs/algorithms, but too slow to be usable

- Theory will help you understand and recognise these

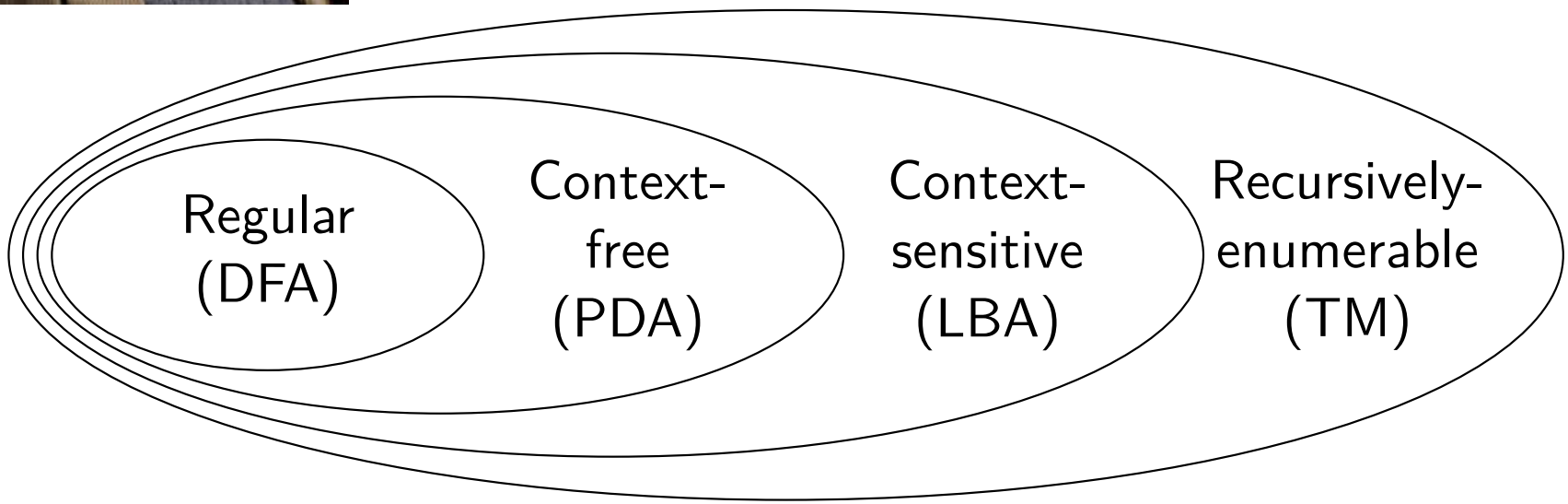
Alan Turing

- Founder of computer science, mathematician, philosopher, code breaker, visionary
- Created abstract machines called *Turing machines* in the 1930s, before computers existed
- Turing test
  - Can machines think?
  - The Imitation Game
- Enigma code breaker

Noam Chomsky



- American linguist, philosopher, cognitive scientist, logician, historian, political critic and activist
- Chomsky Hierarchy: a containment hierarchy of classes of formal languages (applies to human language and computer theory)



Outline

- Why study formal languages and logic?
- **Mathematical notions and definitions**
- Finite State Automata / Regular Languages

Set

Set: An unordered group of unique objects

- *unordered*: $A = \{a, b, c\} = \{b, c, a\} = \{c, b, a\}$
- *contains*: $a \in A$ denotes that a is in A
- *subsets*: if $X \subseteq A$ then every element $x \in X$ is also in A
  - $\{a, c\} \subseteq A$
  - $\{b\} \subseteq A$
  - $\{a, d\} \not\subseteq A$
- *size*: $|A|$ denotes the number of objects in A
- The *empty set* contains no elements (denoted $\emptyset$ or $\{\}$)

2-Tuple

Pair or *2-Tuple*: an ordered list of two objects

- *ordered*: $(a, b) \neq (b, a)$
- *duplicates allowed*: (a, a) is allowed

Set Operations

- *Union*
  - Denoted $A \cup B$
  - The set of elements belonging to at least one of A or B
  - $x \in A \cup B$ if and only if $x \in A$ or $x \in B$ (or both)

Set Operations

- *Union*
  - Denoted $A \cup B$
  - The set of elements belonging to at least one of A or B
  - $x \in A \cup B$ if and only if $x \in A$ or $x \in B$ (or both)
- *Intersection*
  - Denoted $A \cap B$
  - The set of elements belonging to both A and B
  - $x \in A \cap B$ if and only if $x \in A$ and $x \in B$

Set Operations

- *Union*
  - Denoted $A \cup B$
  - The set of elements belonging to at least one of A or B
  - $x \in A \cup B$ if and only if $x \in A$ or $x \in B$ (or both)
- *Intersection*
  - Denoted $A \cap B$
  - The set of elements belonging to both A and B
  - $x \in A \cap B$ if and only if $x \in A$ and $x \in B$
- *Subtraction*
  - Denoted $A \setminus B$
  - The set of elements belonging to A which do not belong to B
  - $x \in A \setminus B$ if and only if $x \in A$ and not $x \in B$

Power Set

$\mathcal{P}(A)$ denotes the Power Set of A, which is the set of all the subsets of A.

- Including $\emptyset$, the *empty set*

Examples:

- If $A = \{0, 1\}$ then $\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$
- If $A = \{a, b, c\}$ then
  $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Cartesian product of two sets A and B

- The set of all pairs where the first element is in A and the second element is in B, denoted $A \times B$
- $(x, y) \in A \times B$ if and only if $x \in A$ and $y \in B$
- $\{0, 1\} \times \{a, b\} = \{(0, a), (0, b), (1, a), (1, b)\}$

Function $f : A \to B$

- Defines an input-output relationship from the set A to the set B
- The set of possible inputs to f is called the domain D, $D \subseteq A$
- The set of possible outputs is called the range R, $R \subseteq B$
- For each $a \in D$, f produces exactly one output $f(a) \in R$.
  - multiple inputs can produce the same output
  - i.e. a *many-to-one* function
  - e.g. if $A = \{0, 1, 2\}, B = \{a, b\}$, then we might define $f : A \to B$ as $f(0) = f(1) = a$ and $f(2) = b$

Alphabet

An *alphabet* is a finite, non-empty set of symbols, denoted Σ

For example:

- Binary: $\Sigma = \{0, 1\}$
- All lower case letters: $\Sigma = \{a, b, c, ..., z\}$
- Alphanumerics: $\Sigma = \{a, ..., z, A, ..., Z, 0, ..., 9\}$
- ASCII, unicode
- Set of signals used by a protocol

String

- A *string* is a finite sequence of symbols from Σ. For example "01110", "alice", "Bnode.java"
- The *length* of a string w, denoted $|w|$, is equal to the number of symbols in the string. e.g. if $x = abcd$ then $|x| = 4$
- Σ^* denotes the set of all strings over the alphabet Σ
- ε (epsilon) denotes the *empty string*
  - $|\varepsilon| = 0$
  - if $x = ab\varepsilon c\varepsilon d\varepsilon$ then $|x| =$?
- $xy =$ the concatenation of two strings x and y
- x^n the string x repeated n times

Powers of an alphabet

Let Σ be an alphabet, then:

- Σ^k is the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$ (i.e. all strings, including ε)
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$ (i.e. all strings except ε)

Languages

- A *language* L over alphabet Σ is a subset of Σ^* (i.e. $L \subseteq \Sigma^*$)
- Examples:
  - Let L be the language of all strings consisting of n 0s followed by n 1s:
    - $L = \{\varepsilon, 01, 0011, 000111, ...\}$
    - $L = \{0^n 1^n | n \geq 0\}$
  - Let L be the language of all strings with an equal number of 0s and 1s:
    - $L = \{\varepsilon, 01, 10, 0011, 0101, 0110, 1001, 1010, 1100, ...\}$
- $\emptyset$ denotes the empty language
  - Beware: $\{\varepsilon\}$ is NOT $\emptyset$

The Membership problem

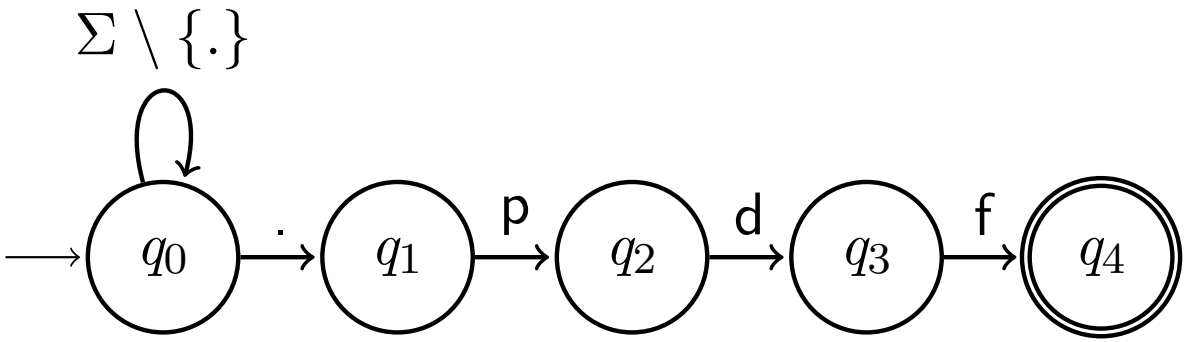
Problem:
Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$

Example:
Let $w = 100011$
Does w belong to the language of strings with an equal number of 0s and 1s?

Outline

- Why study formal languages and logic?
- Mathematical notions and definitions
- **Finite State Automata / Regular Languages**

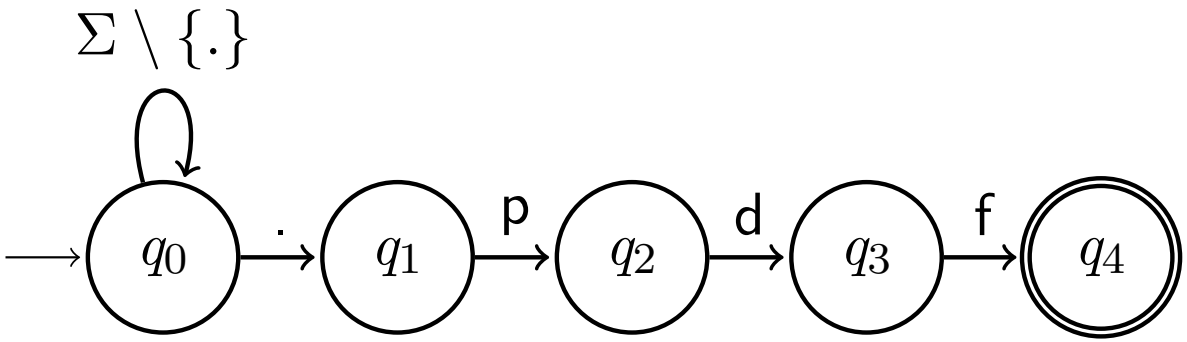
Introductory Example



Formal model which remembers only a finite amount of information

- Information is represented by the *state* (circles)
- *Transition* rules (arrows) define state changes according to input
- *Start* state is denoted $\rightarrow$
- *Accept* state(s) denoted by double circle
- The set of all possible input symbols is the *alphabet*, Σ

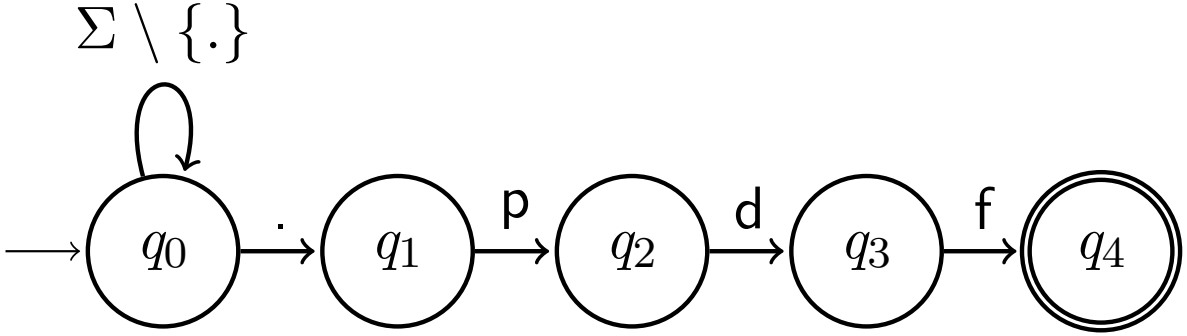
Introductory Example



Given a sequence of input symbols:

- Begin in the *start* state
- Follow one transition for each symbol in the input string
- After reading the entire input string:
  - The input is *accepted* if the automaton is in an *accept* state
  - The input is *rejected* if it is not

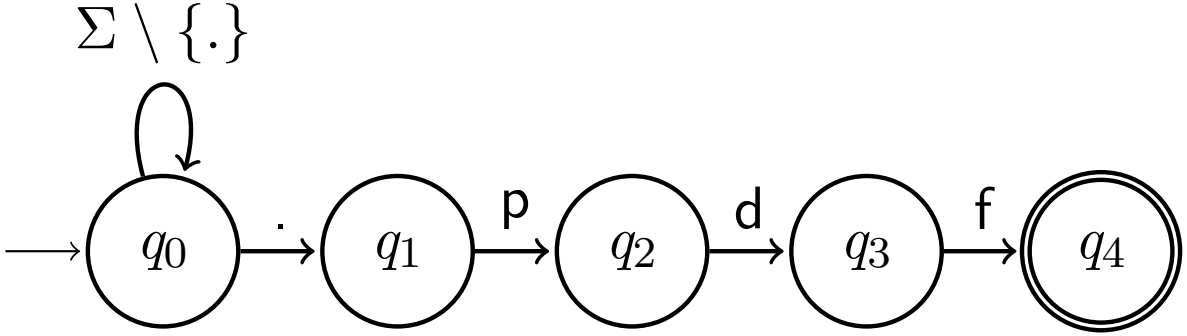
Introductory Example



Example input "example.pdf":

- loops on q_0 through "example"
- moves to q_1 when it scans '.'
- then q_2 for 'p', q_3 for 'd', q_4 for 'f'
- after all input is scanned, we ended in an *accept* state, therefore "example.pdf" is accepted

Introductory Example



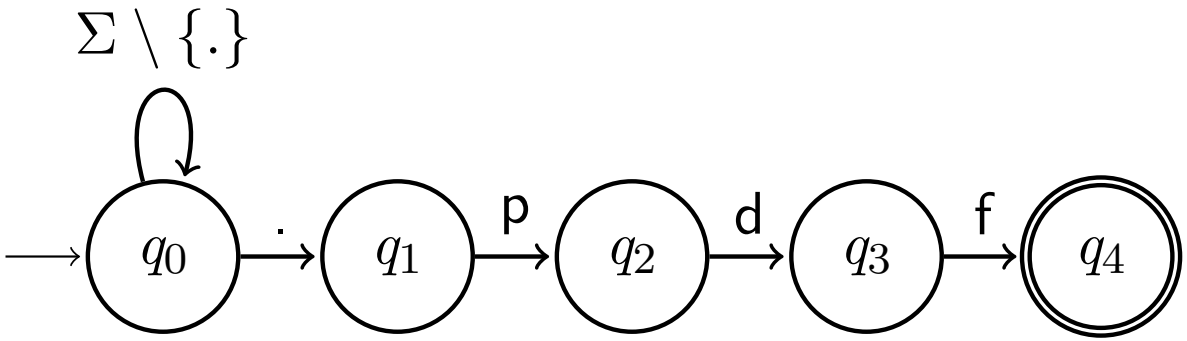
Example input "example.pdf":

- loops on q_0 through "example"
- moves to q_1 when it scans '.'
- then q_2 for 'p', q_3 for 'd', q_4 for 'f'
- after all input is scanned, we ended in an *accept* state, therefore "example.pdf" is accepted

How about:

- .pdf
- pdf
- example.pd
- example.pdf.pdf

Introductory Example



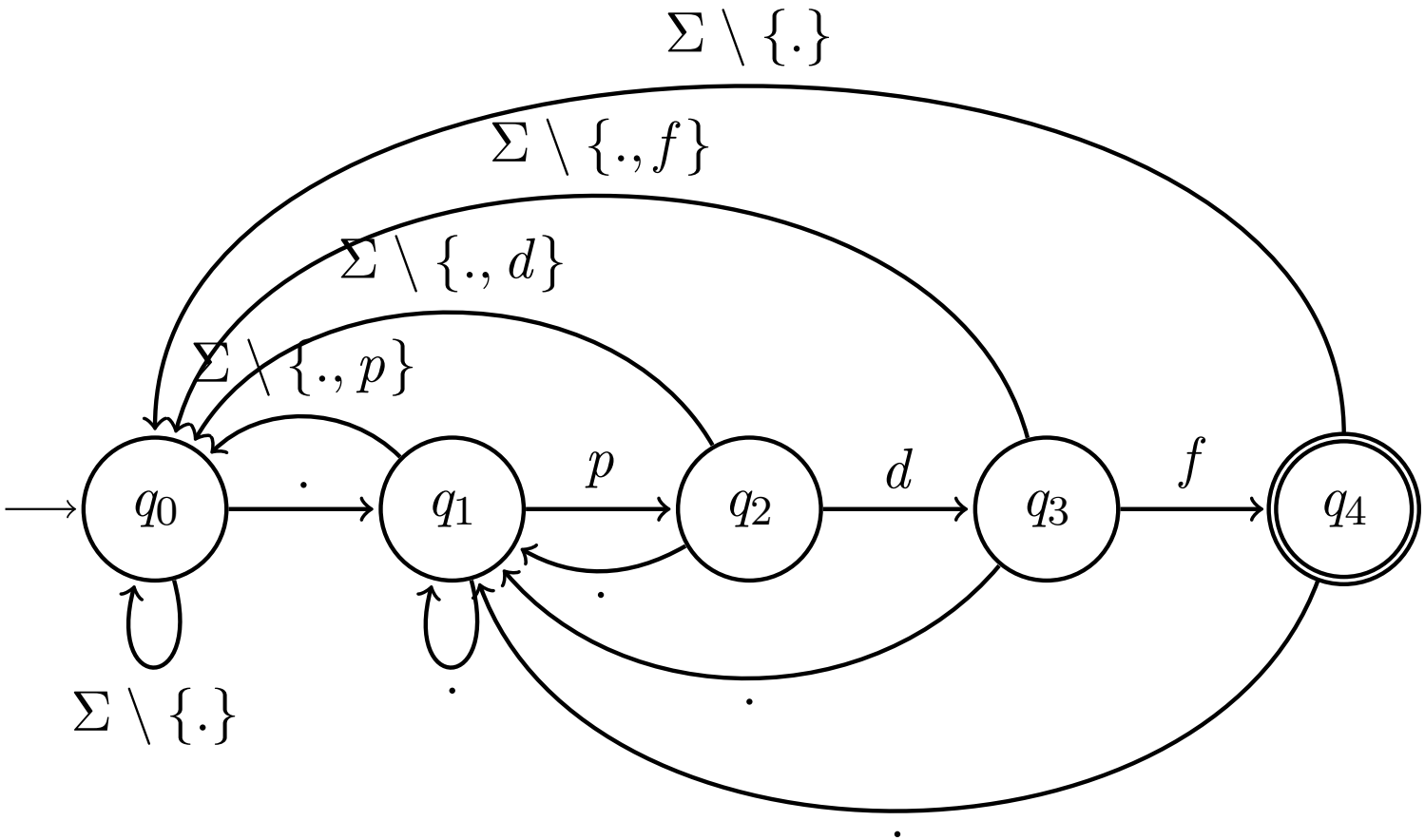
Recall:

- Information is represented by the *state* (circles)

What information is represented by state q_1? By q_2?

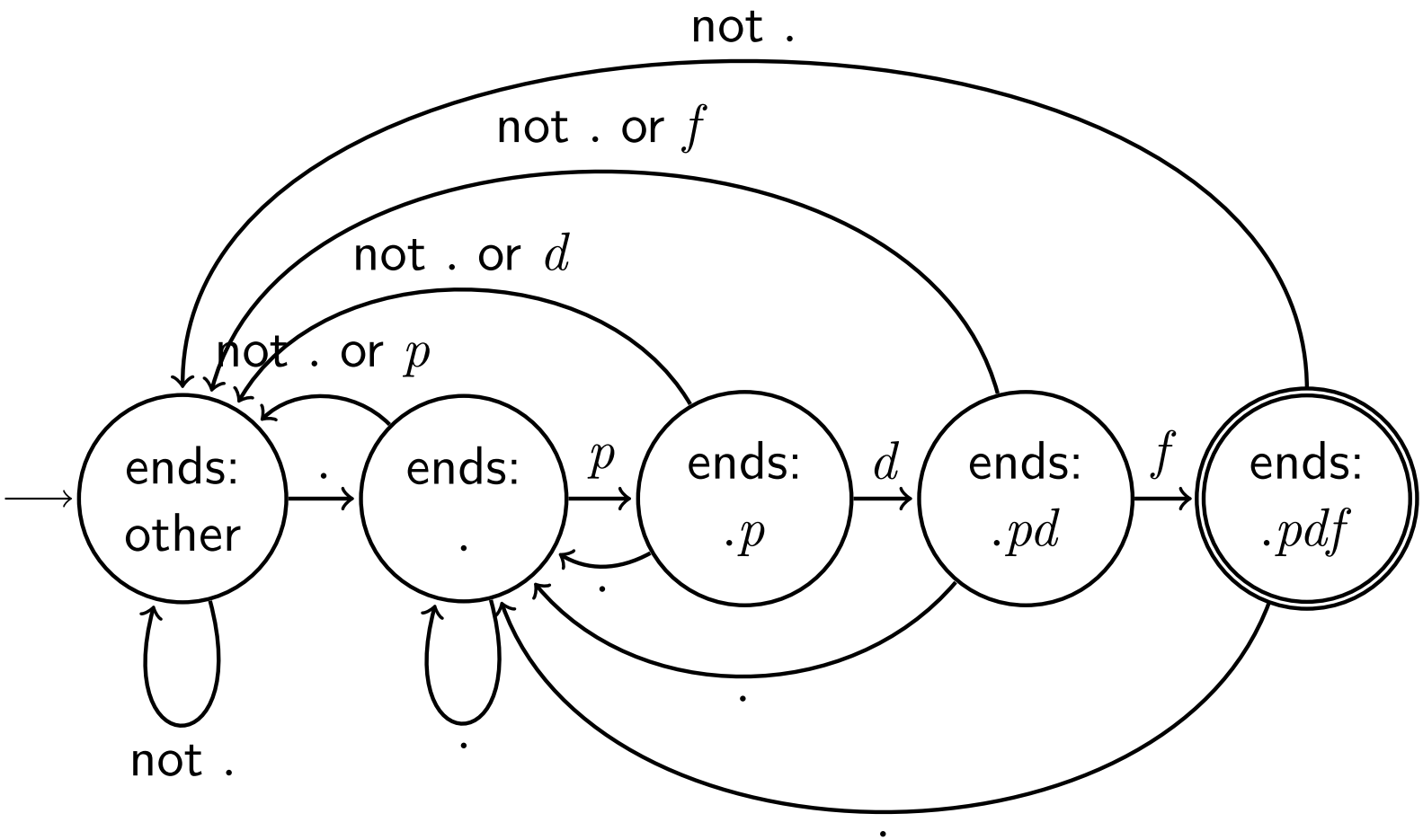
Our first valid DFA

A properly defined DFA will have a transition for every symbol, from every state:



Alternative notation

You can be more descriptive if you like:



Deterministic Finite Automata (DFA)

Informal definition:

1. They have a finite set of *states*
2. They have a finite *alphabet* of symbols
3. They have one *start state*
4. They have a behaviour given by *transitions*
   - Exactly one for each alphabet symbol, from each state
5. They have *accept state(s)*

Deterministic Finite Automata (DFA)

Formal definition:

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

Draw a DFA from the definition

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ where:

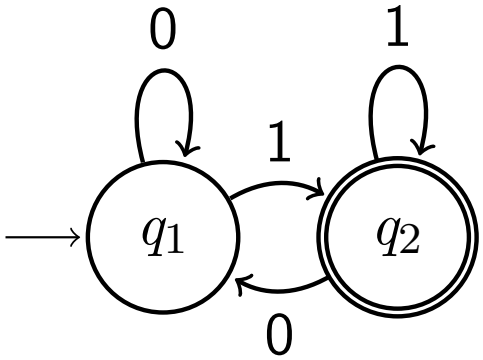
1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1\}$
3. $\delta : Q \times \Sigma \to Q$ is given by

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

4. $q_0 \in Q$ is the start state
5. $F = \{q_1\}$

Now we can draw M_1:

Define a DFA from a diagram



We can formally describe M_2 as:

1. $Q =$
2. $\Sigma =$
3. $\delta : Q \times \Sigma \to Q$ is given by:

4. The start state is:
5. The set of accept states is: $F =$

Some strings where M_2 ends on an accept state are:

Formal definition of computation

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 over the alphabet Σ
Then M accepts w if there exists a sequence of states $r_0 r_1 \cdots r_n$ from Q which satisfy the following three conditions:

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

Language of an automaton

- Automata of all kinds define languages
- Let A be a language, and M be an automaton.
- We say that M *recognises* A if and only if

$$A = \{w | M \text{ accepts } w\}$$

- We often denote the language recognised by M as $L(M)$
- $L(M)$ is the set of all strings labelling paths from the start state of M to any accept state in M

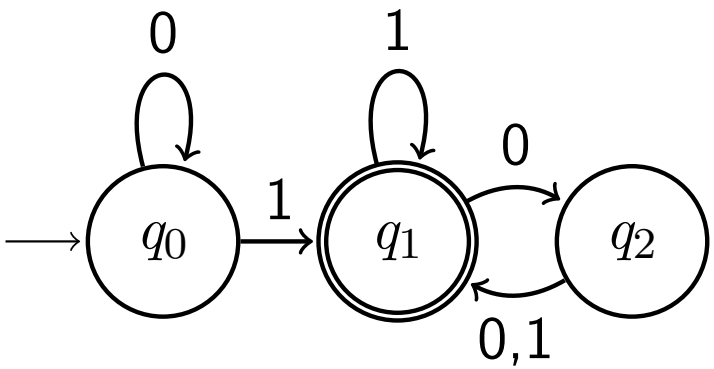
Regular Language

A language is *regular* if and only if there exists a finite automaton which recognises it.

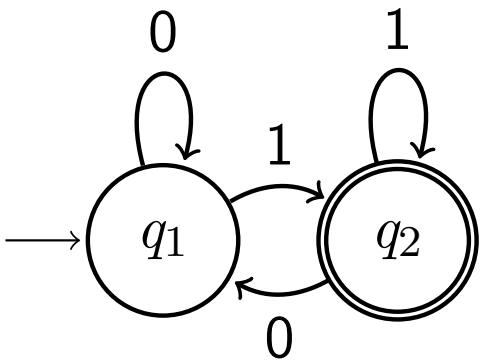
Exercise:
Prove that $A = \{w | w$ is the empty string or ends with a $0\}$ is a regular language

Examples of Regular Languages

What language is accepted by M_1?



Examples of Regular Languages



What language is accepted by M_2?

What is the language recognised by the automaton obtained by inversing the accept and start states in M_2?

Designing finite automata

"Reader as Automaton" method: The *states* encode what you need to *remember* about the string as you are reading it.

Example: Let $\Sigma = \{0, 1\}$. Devise an automaton which accepts the language consisting of strings with an odd number of 1s.

We need to remember:

- There has been an even number of 1s so far
- There has been an odd number of 1s so far

So we will need exactly two states. Then it just remains to add the transitions and define the start and accept states.

Example 2

Devise an automaton which accepts the language of strings which begin and end with 1, over $\{0, 1\}$
We need to remember:

- Whether or not the string began with 1
- Whether or not the last character scanned was a 1

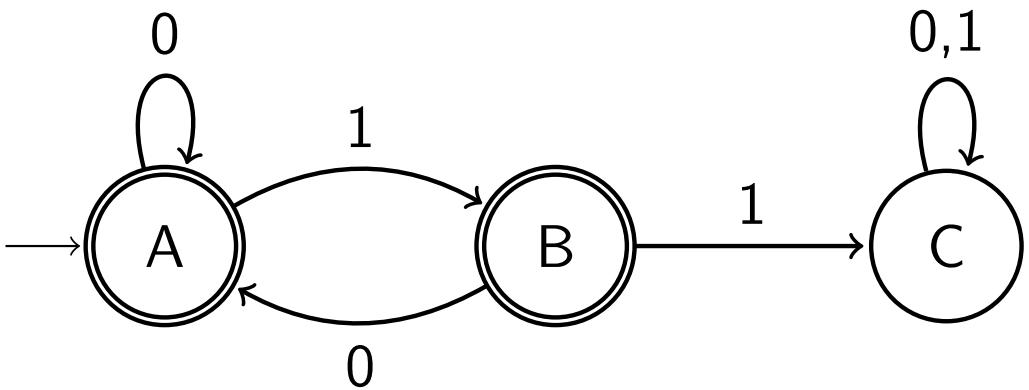
Example 3

Devise an automaton which accepts the language of binary strings which do not contain 11

What do we need to remember?

String in a Language

Let $L = \{w | w \in \{0,1\}^*$ and w does not contain consecutive 1s$\}$
The following DFA accepts the language L



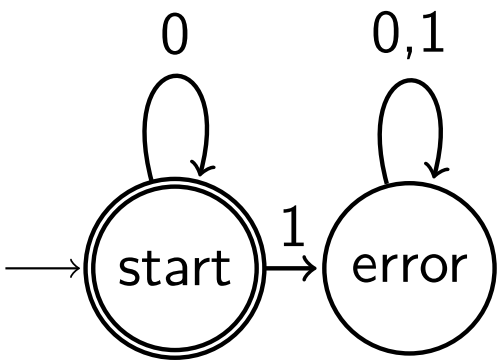
Is the string 101 in L?

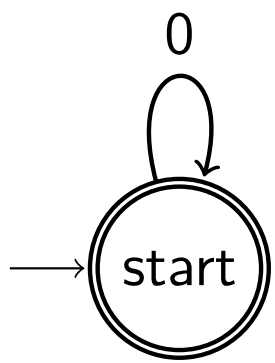
Error State(s)

An *error state* is any state from which it is impossible to reach an accept state. Sometimes we omit these from the diagram. All missing transitions point to an unseen error state.

You *must* write "error states not shown" if you omit them.

These DFA are equivalent and both have *two* states





(error states not shown)

???

With only one state, q_0, how many different finite automata can you devise?

Non deterministic Finite Automata (NFA)

DFA

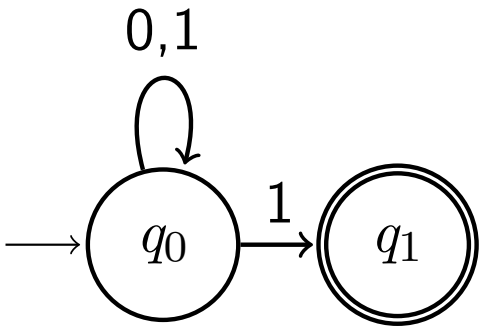
- has exactly one transition per input from each state
- no choice in the computation $\Longrightarrow$ *deterministic*

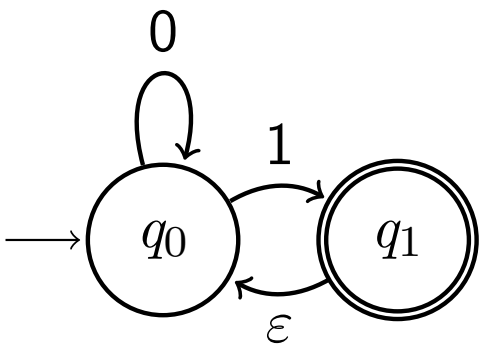
NFA

- can have any number of transitions per input from each state
- so some steps of the computation might be *non-deterministic*
- can also have ε-transitions, i.e. transitions which the automaton can follow without scanning any input

Non deterministic Finite Automata (NFA)

Two examples of NFA which accept the language:
$L(M) = \{w | w \in \{0,1\}^*$ and w ends in $1\}$





Review

- Motivation for studying theory
- Mathematical notions and notation
- DFA
  - Formal definition
  - DFA diagrams
  - Using a DFA to do pattern matching
  - Language of a DFA
  - Building a DFA
- Regular languages
- Definition
- How to prove that a language is regular
- Introduction to NFA