Computer Science 313

Introduction to Deterministic Finite Automata

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #2

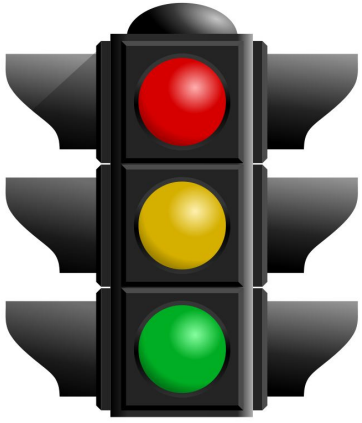
Goals for Today

Goal for Today:

- Introduction of ***deterministic finite automata*** — the simple (abstract) computational devices that will be considered at the beginning of this course

Finite State Machines — An Example

Consider how a simple ***traffic light*** works. For now, let's just worry about the light being shone in *one* direction.



Suppose this light receives (as input) a series of *tick*'s from a timer.

Finite State Machines — An Example

At any point of time (that is, after any input string of *tick*'s has been received and processed), the traffic light is in exactly one of three ***states***:

- q_r: The traffic light is currently *red*.
- q_g: The traffic light is currently *green*.
- q_y: The traffic light is currently *yellow*.

Let us suppose that the traffic light starts by shining *red* — so that its ***initial state*** is the state q_r.

In other words, if the traffic light has not received any *tick*'s yet, at all, then the light is red.

Finite State Machines — An Example

Suppose that we are interested in the input sequences of *tick*'s whose processing allow traffic to move — that is, whose processing cause the traffic light to be in state q_g.

Then q_g should be an ***accepting state***, and q_r and q_y should not — because the sequence of *tick*'s received only allows traffic to move when the light is green.[1]

[1] Yes, I know: In Calgary, drivers also go through traffic lights that are both yellow and red. Let's ignore that detail and continue with the example!

Finite State Machines — An Example

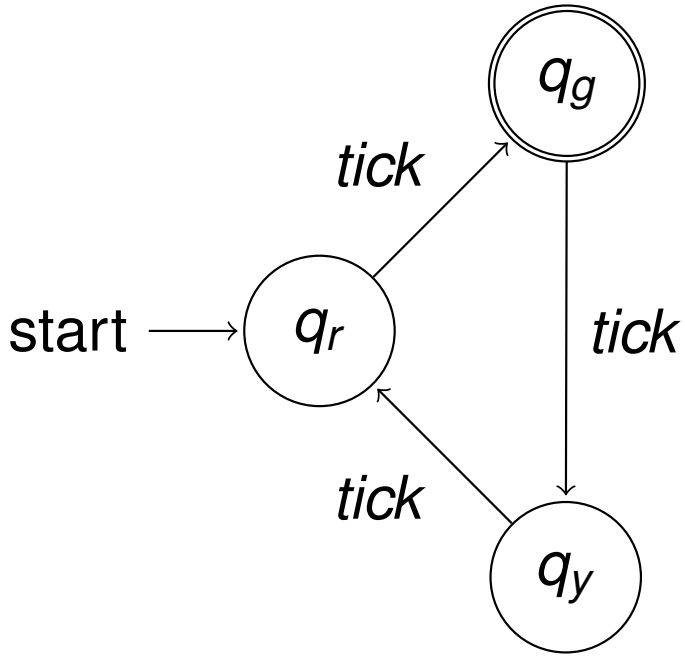
Note that, whenever another *tick* is received and processed, the traffic light ***changes state*** in a well-defined way:

- If the traffic light was in state q_r before this, then it is in state q_g after.
- If the traffic light was in state q_g before this, then it is in state q_y after.
- If the traffic light was in state q_y before this, then it is in state q_r after.

Simplifying Assumption: State changes are assumed to be ***instantaneous***, that is, they do not require any time at all.

Finite State Machines — An Example

The traffic light's behaviour might be represented as follows.



A double circle is used to draw the state q_g in this picture because it is an accepting state.

Fundamental Definitions: Alphabets

We will need to define several other things first before formally defining this kind of finite state machine:

An ***alphabet*** is a finite nonempty set. In this course alphabets will often be named Σ, Σ_i (for an integer $i \geq 0$) and, later on, Γ or Γ_i.

We generally specify an alphabet just by listing its elements — enclosed by curly brackets and separated by commas.

Examples:

- $\Sigma_0 = \{tick\}$
- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{a, b, c\}$

Confession: The first example, above is somewhat unusual: Usually, in this course, the elements of an alphabet will individual symbols (often letters or digits).

Fundamental Definitions: Strings

A ***string over an alphabet*** is a finite *sequence* of symbols from that alphabet, usually written next to each other and *not* separated by commas.

Examples

- If $\Sigma_0 = \{tick\}$ then *tick tick tick* is a string over the alphabet Σ_0.
- If $\Sigma_1 = \{0, 1\}$ then
  - 0
  - 1
  - 00
  - 01
  - 000011010100

  are all strings over Σ_1. Recall that a *sequence* is different from a *set* — so that 0 and 00 are *different* strings over Σ_1 — as are 01 and 10.

Fundamental Definitions: Strings

If ω is a string over an alphabet Σ then the ***length*** of ω is the same as $\omega's$ length as a "sequence." This is usually written as $|\omega|$.

Examples: If $\Sigma_1 = \{0, 1\}$ then the lengths of various strings over Σ_1 are as shown below.

- $|0| = 1$ and $|1| = 1$, as well.
- $|00| = 2$ and $|01| = 2$ as well.
- $|000011010100| = 12$.

Fundamental Definitions: Strings

The ***empty string*** over an alphabet Σ is the unique string over Σ whose length is zero — that is, the "empty sequence."

- In *Introduction to the Theory of Computation,* the empty string over an alphabet Σ is written as ε.
- In other references — and all material provided for CPSC 313 — the empty string over an alphabet Σ will be written as λ instead.

Explanation: ε is just one way to write the (lower case) Greek letter, "epsilon." However, this letter can also be written as $\in$ — which is used in mathematics to mean "is a member of." Using the Greek letter "epsilon" twice, in this way, can be confusing.

Fundamental Definitions: Languages

A ***language over an alphabet Σ*** is a *set* of strings over Σ. Sometimes we will just say that this is a ***language*** (because it will be clear, from context, which alphabet is being used).

Special Cases of Languages — and Notation for Them

- $\emptyset$ — the empty language, that is, the language over the alphabet Σ that does not include any strings at all, so that this is just the "empty set."
- $\Sigma^\star$ — the language over Σ that includes *all* strings over Σ

In material provided for CPSC 313, upper case letters (sometimes with numeral subscripts) — like A, B, C, A_1, A_2 or A_3 — will often be used as names for languages.

Fundamental Definitions: Languages

If a language is *finite* (as a set) then we can specify it just by listing its elements using standard set notation.

Example: If $\Sigma_2 = \{a, b, c\}$ then

$$A = \{aa, ab, cc, acb\}$$

is a language over Σ_2 that includes exactly four strings, namely, the strings *aa*, *ab*, *cc*, and *acb*.

Note: Please remember that languages are *sets* — the order in which you list the elements of a language does not matter.

Fundamental Definitions: Languages

Set notation (or "set-theoretic notation") can also be used to describe languages.

Example: If $\Sigma_2 = \{a, b, c\}$ then we might write

$$B = \{\omega \in \Sigma_2^\star \mid \omega \text{ includes at least two } a\text{'s}\}$$

to say that B includes all the strings ω in $\Sigma_2^\star$ that include at least two copies of the symbol "a" — and that B *does not* include any *other* strings.

Assumption: Students in this course should already be familiar with the above kind of notations to describe sets!

Fundamental Definitions: Languages

Please ***do not*** try to specify a language by giving a few of its elements, and expecting a reader to recognize and apply a pattern!

Explanation for This Request: This is ambiguous, because there will often be *lots* of patterns that can be recognized and used to make up other hypothetical elements of the language being described!

Example: Suppose you are trying to describe a language over the alphabet $\Sigma_2 = \{a, b, c\}$ and you write something like

$$C = \{\lambda, a, aa, aaaa, \dots\}$$

Question: What strings does C include?

Fundamental Definitions: Languages

Possible Answers...

- All strings consisting only of *a*'s whose length is zero or a power of two
- All strings consisting only of *a*'s whose length is either one or even
- All strings consisting only of *a*'s (because there was a typo, and "*aaa*" should also be listed
- All strings in $\Sigma_2^\star$ whose length is zero or a power of two (so that a longer listing would have included *b*, *d*, *ab*, and *ac*)

Please ***do not*** assume that another reader (or marker) will 'fill in the blanks" or "extend a pattern" the same way you would, on assignments and tests in this course.

Why Do We Care about Strings and Languages?

These things are "abstractions" or "generalizations" of more concrete things concerning computation:

- An ***alphabet*** generalizes the set of things that can be stored using a single word of computer memory.
- A ***string*** corresponds to an "encoding" of an instance of a problem to be solved, that is, what a computer "really works with."
- A ***language*** corresponds to a set of inputs having a useful property you want to recognize — so you want to be able to use a machine or algorithm to distinguish between the strings that are *in* the language, and the strings that are *not in* the language.

Fundamental Definitions: Decision Problems

A ***decision problem*** is a computational problem for which the output is always either "Yes" or "No."

Every ***alphabet*** Σ and ***language*** L over Σ defines a ***decision problem*** such that

- The set of valid ***instances*** (or "inputs") for the problem is the set $\Sigma^\star$ of all strings over the alphabet Σ.
- The set of "Yes-instances," that is, instances for which the correct output is "Yes," is the language L.
- The set of "No-instances," that is, instances for which the correct output is "No," is the set of strings in $\Sigma^\star$ that *do not* belong to L.

This kind of computational problem will be studied for the rest of CPSC 313.

Formal Definition

As described above a traffic light is an example of a ***deterministic finite automaton***. This is formally (i.e. mathematically) described as follows.

Definition: A ***deterministic finite automaton (DFA)*** is (formally modelled as) a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite, nonempty, set of ***states***;
- Σ is an ***alphabet*** (as previously defined) — we normally require that $Q \cap \Sigma = \emptyset$;
- $\delta : Q \times \Sigma \to Q$ is a ***transition function***;
- $q_0 \in Q$ is the ***start state***; and
- $F \subseteq Q$ is the set of ***accept states***.

DFA's — Example Continued

Example: The above *traffic light* can be modelled as a deterministic finite automaton

$$M = (Q, \Sigma, \delta, q_r, F)$$

where

- $Q = \{q_r, q_g, q_y\}$;
- $\Sigma = \{tick\}$ — that is, the alphabet previously called Σ_0;
- δ is as shown on the following slide;
- q_r is the initial state (as shown by the 5-tuple given above as M); and
- $F = \{q_g\}$ — so that q_g is the only accepting state.

DFA's — Example Continued

In general, the ***transition function*** $\delta : Q \times \Sigma \to Q$ is a *total* function from the set of ordered pairs of states and elements of the alphabet to the set of states. For $q, r \in Q$ and $\sigma \in \Sigma$,

$$\delta(q, \sigma) = r$$

if and only if the following holds: If the automaton is in state q and the symbol σ is received, then the automaton moves to state r.

Thus, in the *traffic light* example,

$$\delta(q_r, \mathit{tick}) = q_g, \quad \delta(q_g, \mathit{tick}) = q_y, \quad \text{and} \quad \delta(q_y, \mathit{tick}) = q_r.$$

DFA's — Example Continued

A transition function can be shown by a ***transition table*** in which

- rows are indexed by *states* in Q,
- columns are indexed by *symbols* in Σ, and
- for every state $q \in Q$ and symbol $\sigma \in \Sigma$, the state r in row q and column σ of the table is $\delta(q, r)$.

Thus the transition function for the *traffic light* is given by the following transition table:

	tick
q_r	q_g
q_g	q_y
q_y	q_r

Extended Transition Functions

The ***extended transition function*** is a total function

$$\delta^\star : Q \times \Sigma^\star \to Q$$

such that if the automaton M is in state $q \in Q$, and the (sequence of symbols in) the string $\omega \in \Sigma^\star$ is received, then M is in state $\delta^\star(q, \omega)$ after that.

Thus

$$\delta^\star(q, \lambda) = q$$

for every state $q \in Q$.

Extended Transition Functions

Suppose that

$$\omega = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^\star$$

is a string with length $n \geq 0$ in $\Sigma^\star$ — so that, for $1 \leq i \leq n$, the i^{th} symbol in ω is σ_i.

Let $q \in Q$. Then a *sequence* of states $r_0, r_1, \ldots, r_n$, with length $n+1$, can be defined as follows:

- $r_0 = q$, and
- $r_{i+1} = \delta(r_i, \sigma_{i+1})$ for every integer i such that $0 \leq i < n$.

Easily Proved by Induction on i: If the automaton M is in state q, and the string $\sigma_1\sigma_2 \ldots \sigma_i$ is received, then M is in state r_i immediately after that.

Thus $\delta^\star(q, \omega) = r_n$, that is, $\delta^\star(q, \omega)$ is the *last* state in the above sequence.

Extended Transition Functions

Suppose

$$\mu = \sigma_1\sigma_2\ldots\sigma_n \quad \text{and} \quad \nu = \tau_1\tau_2\ldots\tau_m$$

are strings in $\Sigma^\star$ with lengths n and m respectively. Then the ***concatenation*** $\mu \cdot \nu$ of μ and ν is the string

$$\sigma_1\sigma_2\ldots\sigma_n\tau_1\tau_2\ldots\tau_m$$

with length $n + m$ that is produced by writing the symbols in ν immediately after the symbols in μ. Similarly, if

$$\mu = \sigma_1\sigma_2\ldots\sigma_n$$

is a string with length n in $\Sigma^\star$, and τ is a symbol in Σ, then the ***concatenation*** $\mu \cdot \tau$ of μ and τ is the string

$$\sigma_1\sigma_2\ldots\sigma_n\tau$$

with length $n + 1$ produced by writing τ after the symbols in μ.

Extended Transition Function

Useful Observation: If $\omega = \sigma_1\sigma_2\ldots\sigma_n$ is a string with length $n \geq 1$ in $\Sigma^\star$ then

$$\omega = \mu \cdot \tau$$

for the string

$$\mu = \sigma_1\sigma_2\ldots\sigma_{n-1}$$

with length $n-1$ in $\Sigma^\star$ and for the symbol $\tau = \sigma_n \in \Sigma$.

Another Equivalent Definition of the Extended Transition Function: For every state $q \in Q$ and string $\omega \in \Sigma^\star$,

$$\delta^\star(q,\omega) = \begin{cases} q & \text{if } \omega = \lambda, \\ \delta(\delta^\star(q,\mu),\tau) & \text{if } \omega = \mu\cdot\tau \text{ for } \mu \in \Sigma^\star \text{ and } \tau \in \Sigma. \end{cases}$$

A ***proof*** of the equivalence of these definitions is given in the supplemental document for this lecture.

Extended Application Function

Application of Second Definition: Evaluation of the function!

Example:

$$\begin{aligned}
\delta^\star(q_r, \mathit{tick}\ \ \mathit{tick}\ \ \mathit{tick}) &= \delta(\delta^\star(q_r, \mathit{tick}\ \ \mathit{tick}), \mathit{tick}) \\
&= \delta(\delta(\delta^\star(q_r, \mathit{tick}), \mathit{tick}), \mathit{tick}) \\
&= \delta(\delta(\delta(\delta^\star(q_r, \lambda), \mathit{tick}), \mathit{tick}), \mathit{tick}) \\
&= \delta(\delta(\delta(q_r, \mathit{tick}), \mathit{tick}), \mathit{tick}) \\
&= \delta(\delta(q_g, \mathit{tick}), \mathit{tick}) \\
&= \delta(q_y, \mathit{tick}) \\
&= q_r.
\end{aligned}$$

In the first three lines, the *second* part of the definition is used (with various choices of ω, μ and τ). The *first* part of the definition is used in the line after that, and then the definition of the (regular) transition function δ is repeatedly used to finish the evaluation.

The Language of an Automaton

Suppose again that M is the deterministic finite automaton

$$M = (Q, \Sigma, \delta, q_0, F).$$

Then, for every string $\omega \in \Sigma^\star$, M ***accepts*** ω if and only if

$$\delta^\star(q_0, \omega) \in F,$$

and M ***rejects*** ω otherwise.

The ***language*** $L(M)$ of an automaton M is the set of strings $\omega \in \Sigma^\star$ such that M accepts ω.

The Language of an Automaton

For the automaton M modelling the *traffic light*, described above, M accepts the string

$$\omega_i = \mathit{tick}\,\mathit{tick}\,\ldots\,\mathit{tick}$$

with length i if and only if $i = 3 \times k + 1$ for some integer $k \geq 0$ — because

$$\delta^\star(q_r, \omega_i) = q_g$$

if and only if $i = 3 \times k + 1$ for some integer $k \geq 0$.

Thus, if M is the *traffic light* automaton, then $\Sigma = \{\mathit{tick}\}$ and

$$L(M) = \{\omega \in \Sigma^\star \mid |\omega| = 3 \times k + 1 \text{ for an integer } k \geq 0\}.$$

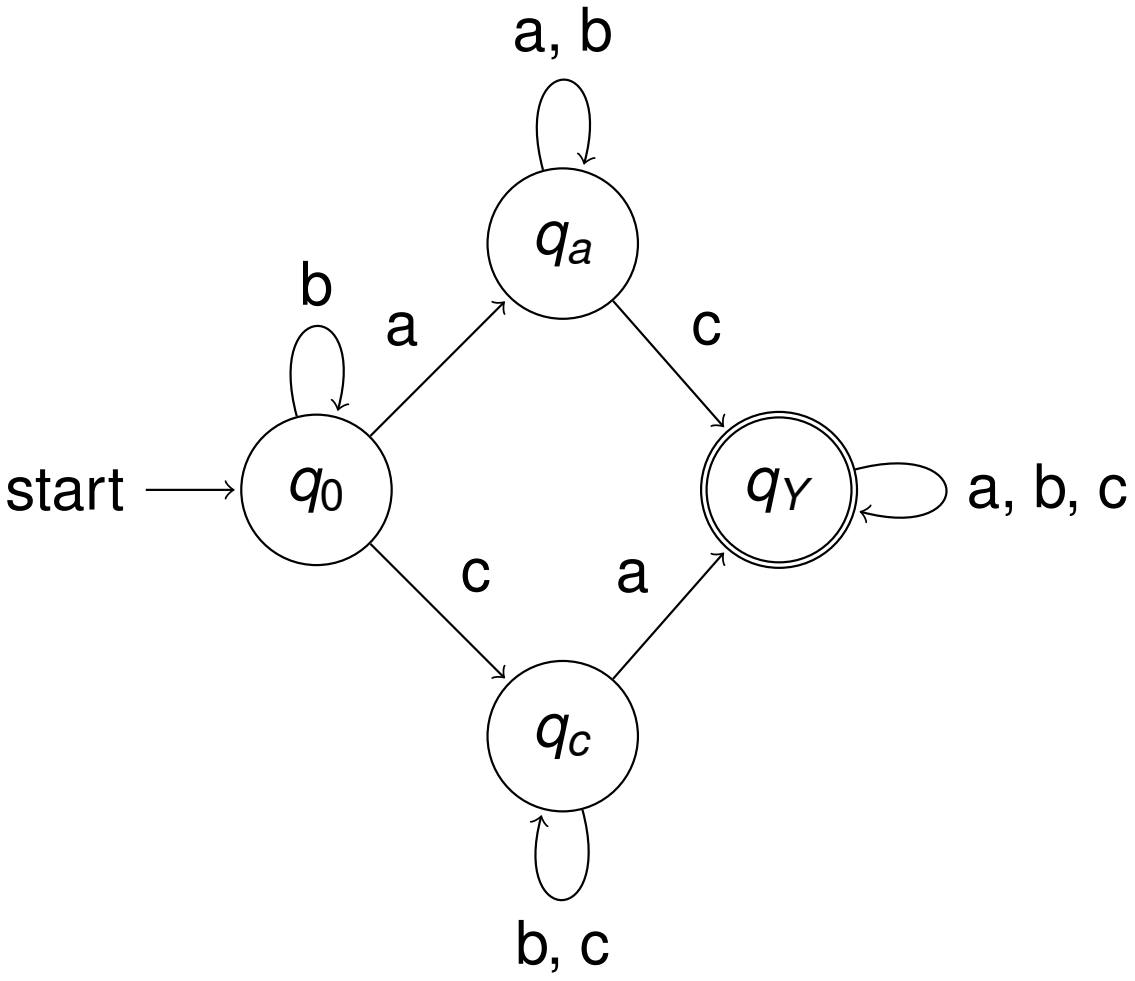
Regular Languages

Definition: For every alphabet Σ, a language $L \subseteq \Sigma^\star$ is a ***regular language*** if (and only if) $L = L(M)$ for some deterministic finite automaton M with alphabet Σ.

Properties and applications of regular languages will be considered at the beginning of this course.

Another Example

Consider the following deterministic automaton, which processes strings over the alphabet $\Sigma = \{a, b, c\}$:



Another Example

This can be modelled as a deterministic finite automaton

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- $Q = \{q_0, q_a, q_c, q_Y\}$;
- $\Sigma = \{a, b, c\}$, as noted above;
- the transition function δ has the transition table shown on the next slide;
- as the above 5-tuple indicates, q_0 is the start state, and
- $F = \{q_Y\}$.

Another Example

The transition function for this deterministic finite automaton has the following transition table.

	a	b	c
q_0	q_a	q_0	q_c
q_a	q_a	q_a	q_Y
q_c	q_Y	q_c	q_c
q_Y	q_Y	q_Y	q_Y

That is,

- $\delta(q_0, a) = q_a$, $\delta(q_0, b) = q_0$, and $\delta(q_0, c) = q_c$;
- $\delta(q_a, a) = q_a$, $\delta(q_a, b) = q_a$, and $\delta(q_a, c) = q_Y$;
- $\delta(q_c, a) = q_Y$, $\delta(q_c, b) = q_c$, and $\delta(q_c, c) = q_c$;
- $\delta(q_Y, a) = q_Y$, $\delta(q_Y, b) = q_Y$, and $\delta(q_Y, c) = q_Y$.

Another Example

Now consider the following sets. Note that every string in $\Sigma^\star$ belongs to ***exactly one*** of these:

- $S_0 = \{\omega \in \Sigma^\star \mid \omega$ does not include any a's or c's$\}$;
- $S_a = \{\omega \in \Sigma^\star \mid \omega$ includes at least one a but no c's$\}$;
- $S_c = \{\omega \in \Sigma^\star \mid \omega$ includes at least one c but no a's$\}$;
- $S_Y = \{\omega \in \Sigma^\star \mid$ ω includes at least one a and at least one c$\}$.

Then for every string $\omega \in \Sigma^\star$...

- ... $\delta^\star(q_0, \omega) = q_0$ if and only if $\omega \in S_0$.
- ... $\delta^\star(q_0, \omega) = q_a$ if and only if $\omega \in S_a$.
- ... $\delta^\star(q_0, \omega) = q_c$ if and only if $\omega \in S_c$.
- ... $\delta^\star(q_0, \omega) = q_Y$ if and only if $\omega \in S_Y$.

Since q_Y is the only accepting state, the language of this automaton is the above set S_Y.

Why are DFA's Important?

- ***Easy Exercise:*** Suppose you are given a DFA whose language, Σ, is a subset of Java or Python *characters*. Write a Python or Java program that takes a string $\omega \in \Sigma^\star$ as input, and reports whether this string is accepted by the given DFA.
- As noted in the preface to *Introduction to the Theory of Computation,* ***regular expressions*** are extremely useful for string searching and pattern matching.

  We will see, by the end of this unit, that DFA's (and "NFA's," which will be defined shortly) are extremely useful when you are using regular expressions.

Additional Material

Additional Examples:

- Section 1.1 — called EXAMPLES OF FINITE AUTOMATA, in *Introduction to the Theory of Computation* — includes another five examples of these machines.

What's Next?

- The next lecture introduced a ***design process*** that can be used to design a deterministic finite automaton with a given language.