Formal Definition of Finite Automaton

1. Finite set of *states*, typically Q.
2. Alphabet of *input symbols*, typically Σ.
3. One state is the *start/initial* state, typically q_0.
4. Zero or more *final/accepting* states; the set is typically F.
5. A *transition function*, typically δ. This function:
   - Takes a state and input symbol as arguments.
   - Returns a state.
   - One "rule" of δ would be written $\delta(q, a) = p$, where q and p are states, and a is an input symbol.
   - Intuitively: if the FA is in state q, and input a is received, then the FA goes to state p (note: $q = p$ OK).

- A FA is represented as the five-tuple: $A = (Q, \Sigma, \delta, q_0, F)$.

Example: Clamping Logic

We may think of an accepting state as representing a "1" output and nonaccepting states as representing "0" out.

A "clamping" circuit waits for a 1 input, and forever after makes a 1 output. However, to avoid clamping on spurious noise, we'll design a FA that waits for two 1's in a row, and "clamps" only then.

In general, we may think of a state as representing a summary of the history of what has been seen on the input so far. The states we need are:

1. State q_0, the start state, says that the most recent input (if there was one) was not a 1, and we have never seen two 1's in a row.
2. State q_1 says we have never seen 11, but the previous input was 1.
3. State q_2 is the only accepting state; it says that we have at some time seen 11.

- Thus, $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$, where δ is given by:

	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_0	q_2
$*q_2$	q_2	q_2

- By marking the start state with $\rightarrow$ and accepting states with $*$, the *transition table* that defines δ also specifies the entire FA.

Conventions

It helps if we can avoid mentioning the type of every name by following some rules:

- Input symbols are a, b, etc., or digits.
- Strings of input symbols are $u, v, \ldots, z$.
- States are q, p, etc.

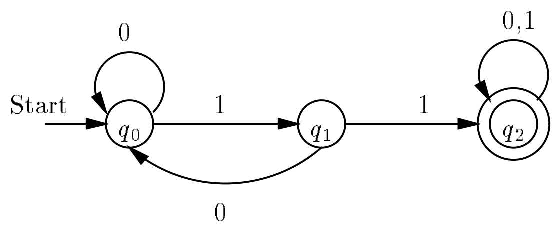
Transition Diagram

A FA can be represented by a graph; nodes = states; arc from q to p is labeled by the set of input symbols a such that $\delta(q, a) = p$.

- No arc if no such a.
- Start state indicated by word "start" and an arrow.
- Accepting states get double circles.

Example

For the clamping FA:



Extension of δ to Paths

Intuitively, a FA *accepts* a string $w = a_1a_2 \cdots a_n$ if there is a path in the transition diagram that:

1. Begins at the start state,
2. Ends at an accepting states, and
3. Has sequence of labels $a_1, a_2, \ldots, a_n$.

Formally, we extend transition function δ to $\hat{\delta}(q, w)$, where w can be any string of input symbols:

- Basis: $\hat{\delta}(q, \epsilon) = q$ (i.e., on no input, the FA doesn't go anywhere.
- Induction: $\hat{\delta}(q, wa) = \delta\big(\hat{\delta}(q, w), a\big)$, where w is a string, and a a single symbol (i.e., see where the FA goes on w, then look for the transition on the last symbol from that state).
- Important fact with a straightforward, inductive proof: $\hat{\delta}$ really represents paths. That is, if $w = a_1 a_2 \cdots a_n$, and $\delta(p_i, a_i) = p_{i+1}$ for all $i = 0, 1, \ldots, n-1$, then $\hat{\delta}(p_0, w) = p_n$.

Acceptance of Strings

A FA $A = (Q, \Sigma, \delta, q_0, F)$ accepts string w if $\hat{\delta}(q_0, w)$ is in F.

Language of a FA

FA A accepts the language $L(A) = \{w \mid \hat{\delta}(q_0, w)$ is in $F\}$.

Aside: Type Errors

A major source of confusion when dealing with automata (or mathematics in general) is making "type errors."

- Example: Don't confuse A, a FA, i.e., a program, with $L(A)$, which is of type "set of strings."
- Example: the start state q_0 is of type "state," but the accepting states F is of type "set of states."
- Trickier example: Is a a symbol or a string of length 1?
  - ❖ Answer: it depends on the context, e.g., is it used in $\delta(q, a)$, where it is a symbol, or $\hat{\delta}(q, a)$, where it is a string?

Nondeterministic Finite Automata

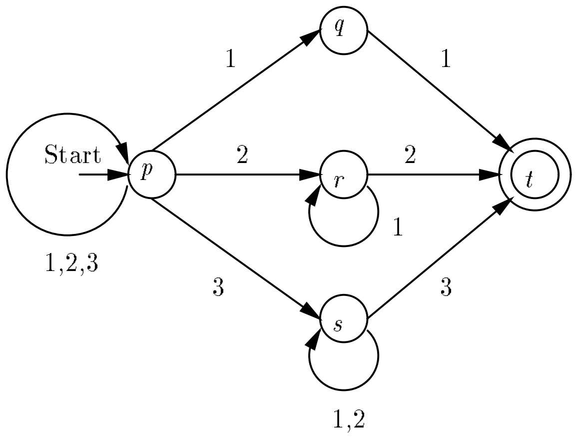
Allow (deterministic) FA to have a choice of 0 or more next states for each state-input pair.

- Important tool for designing string processors, e.g., `grep`, lexical analyzers.
- But "imaginary," in the sense that it has to be implemented deterministically.

Example

In this somewhat contrived example, we shall design an NFA to accept strings over alphabet $\{1, 2, 3\}$ such that the last symbol appears previously, without any intervening higher symbol, e.g., $\cdots 11$, $\cdots 21112$, $\cdots 312123$.

- Trick: use start state to mean "I guess I haven't seen the symbol that matches the ending symbol yet.
- Three other states represent a guess that the matching symbol has been seen, and remembers what that symbol is.



Formal NFA

$N = (Q, \Sigma, \delta, q_0, F)$, where all is as DFA, but:

- $\delta(q, a)$ is a *set* of states, rather than a single state.

Extension to $\hat{\delta}$

- Basis: $\hat{\delta}(q, \epsilon) = \{q\}$.
- Induction: Let:
  - ❖ $\hat{\delta}(q, w) = \{p_1, p_2, \ldots, p_k\}$.
  - ❖ $\delta(p_i, a) = S_i$ for $i = 1, 2 \ldots, k$.

  Then $\hat{\delta}(q, wa) = S_1 \cup S_2 \cup \cdots \cup S_k$.

Language of an NFA

An NFA accepts w if *any* path from the start state to an accepting state is labeled w. Formally:

- $L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

Subset Construction

- For every NFA there is an *equivalent* (accepts the same language) DFA.
- But the DFA can have exponentially many states.

Let $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ be an NFA. The equivalent DFA constructed by the subset construction is $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$, where:

1. $Q_D = 2^{Q_N}$;. i.e., Q_D is the set of all subsets of Q_N.

2. F_N is the set of sets S in Q_D such that $S \cap F \neq \emptyset$.

   $\delta_D(\{q_1, q_2, \ldots, q_k\}, a) = \delta_N(p_1, a) \cup \delta_N(p_2, a) \cup \cdots \cup \delta_N(p_k, a)$.

- Key theorem (induction on $|w|$, proof in book): $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$.

- Consequence: $L(D) = L(N)$.

Example: Subset Construction From Previous NFA

An important practical trick, used in lexical analyzers and other text-processors is to ignore the (often many) states that are not accessible from the start state (i.e., no path leads there).

- For the NFA example above, of the 32 possible subsets, only 15 are accessible. Computing transitions "on demand" gives the following δ_D:

	1	2	3
$\rightarrow p$	pq	pr	ps
pq	pqt	pr	ps
$*pqt$	pqt	pr	ps
pr	pqr	prt	ps
$*prt$	pqr	prt	ps
ps	pqs	prs	pst
$*pst$	pqs	prs	pst
prs	$pqrs$	$prst$	pst
$*prst$	$pqrs$	$prst$	pst
pqs	$pqst$	prs	pst
$*pqst$	$pqst$	prs	pst
pqr	$pqrt$	prt	ps
$*pqrt$	$pqrt$	prt	ps
$pqrs$	$pqrst$	$prst$	pst
$*pqrst$	$pqrst$	$prst$	pst