In the definition of a finite state acceptor, t is the transition mapping, which may or may not be a function (hence the careful terminology). This is because finite state acceptors come in 2 flavours:

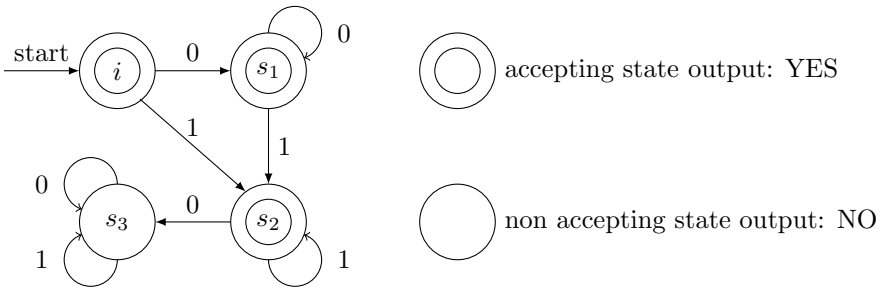
1. Deterministic: every state has exactly one transition for each possible input, **i.e.** $\forall(s,a) \in S \times A \; \exists! \; t(s,a) \in S$. In other words, the transition mapping is a function.
2. Non-deterministic: an input can lead to one, more than one or no transition for a given state. Some $(s,a) \in S \times A$ might be assigned to more than one element of S, **i.e.** the transition mapping is not a function.

Surprisingly $\exists$ algorithm that transforms a non-deterministic (thought more complex one) using the power set construction.
As a result, we have the following theorem:

Theorem: A language L over some alphabet A is a regular language $\Leftrightarrow$ L is recognised by a deterministic finite state acceptor with input alphabet $A \Leftrightarrow$ L is recognised by a non-deterministic finite state acceptor with input alphabet A.

Example: Build a deterministic finite state acceptor for the regular language $L = \{0^m 1^n \mid m,n \in \mathbb{N}, m \geq 0, n \geq 0\}$

Accepting states in this examples: i, s_1, s_2
Non accepting states: s_3
Start states: i
Here $S = \{i, s_1, s_2, s_3\}$ $F = \{i, s_1, s_2\}$ $A = \{0,1\}$ $t : S \times A \rightarrow S$ $t(i,0) = s_1$ $t(i,1) = s_2$ $t(s_1,0) = s_1$ $t(s_1,1) = s_2$ $t(s_2,0) =$



$s_3 \quad t(s_2, 1) = s_2$

Let's process some strings:

String	ε (empty string)
State (i)	i
Output	YES

String	0	0	1	1	1
State i	s_1	s_1	s_2	s_2	s_2
Output	YES				

String	1	1
State i	s_2	s_2
Output	YES	

String	1
State i	s_2
Output	YES

String	0	1	0	1
State i	s_1	s_2	s_3	s_3
Output	NO			

Now that we really understand what a finite state acceptor is, we can develop a criterion for recognising regular languages called the Myhill-Nerode theorem based on an equivalence relation we can set up on words in our language over the alphabet A.

Definition: Let $x, y \in L$, a language over the alphabet A. We call x and y equivalent over L denoted by $x \equiv_L y$ if $\forall w \in A^*, xw \in L \Leftrightarrow yw \in L$.

Note: xw means the concatenation $x \circ w$, and yw is the concatenation $y \circ w$.

Idea: If $x \equiv_L y$, then x and y place our finite state acceptor into the same state s.

Notation: Let L/N be the set of equivalence classes determined by the equivalence relation $\equiv_L$.

The Myhill-Nerode Theorem: Let L be a language over the alphabet A. If the set L/N ofequivalence classes in L is infinite, then L is not a regular language.

Stretch of Proof: All element of one equivalence class in L/N place our automation into the same state s. Elements of distinct equivalence classes place the automation into distinct state, **i.e.** if $[x], [y] \in L/N$ and $[x] \neq [y]$, then all elements of $[x]$ place the automation into some state s, while all

elements of $[y]$ place the automation into some state s', with $s \neq s' \Rightarrow$ an automation that can recognise L has as many states at the number of equivalence classes in L/N, but L/N is NOT finite $\Rightarrow$ L cannot be recognised by a finite state automation $\Rightarrow$ L is not regular by the theorem above.

qed

8.4 Regular Grammars

Task: Understand what is the form of the production rules of a grammar that generates a regular language.

Recall: that a context-free grammar is given by $(V, A, <s>, P)$ where every production rule <T>$\rightarrow w$ in P causes one and only one nonterminal to be replaced by a string in V^*.

Definition: A context-free grammar $(V, A, <s>, P)$ is called a regular grammar is every production rule in P is of one of the three forms:

(i) <A>$\rightarrow$ b<B>

(ii) <A>$\rightarrow$ b

(iii) <A>$\rightarrow \varepsilon$

where <A> and <B> are nonterminals, b is a terminal, and ε is the empty word. A regular grammar is said to be in normal form if all its production rules are of types (i) and (iii).

Remark: In the literature, you often see this definition labelled left-regular grammar as opposed to right-regular grammar, where the production rules of types 1 have the form <A>$\rightarrow$<B>b, (**i.e.** the terminal is one the right of the nonterminal). This distinction is not really important as long as we stick to one type throughout since both left-regular grammars and right-regular grammars generate regular languages.

Lemma: Any language generated by a regular grammar may be generated by a regular grammar in normal form.

Proof: Let <A>$\rightarrow$b be a rule of type (ii). Replace it by two rules: <A>$\rightarrow$b<F> and <F>$\rightarrow \varepsilon$, where <F> is a new nonterminal. Add <F> to the set V. We do the same for every rule of type (ii) obtaining a bigger set V, but now our production rules are only of type (i) and (iii) and we are generating the same language.

qed

Example: Recall the regular language $L = \{0^m 1^n \mid m, n \in \mathbb{N}, m \geq 0, n \geq 0\}$. We can generate it from the regular grammar in normal gorm given by production rules:

1. <s>→ 0<A>
2. <A>→ 0<A>
3. <A>→ ε
4. <s>→ ε
5. <A>→ 1<B>
6. <B>→ 1<B>
7. <s>→ 1 <B>
8. <B>→ ε

Rules (1), (2), (5), (6), (7) are of type (i), where rules (3), (4) and (8) are of types (iii).
(1) and (3) gives 0. (1), (2) applied $m-1$ times and (3) gives 0^m for $m \geq 2$.
(7) and (8) give 1. (7), (6) applied $n-1$ times and (8) give 1^n for $n \geq 2$.
(1), (5) and (8) give 01. (1), (5), (6) applied $n-1$ times and (8) gives 01^n for $n \geq 2$.
(1), (2) applied $m-1$ times, (5) and (8) gives 0^m1 for $m \geq 2$.
(1), (2) applied $m-1$ times, (5), (6) applied $n-1$ times, and (8) gives 0^m1^n for $m \geq 2, n \geq 2$.
Rule (4) gives the empty word $\varepsilon = 0^01^0$.

Q: Why does a regular grammar yield a regular language, **i.e.** one recognised by a finite state acceptor?

A: Not obvious from the definition, <u>but</u> we can construct the finite state acceptor from the regular grammar as follows: our regular grammar is given by $(V, A, <s>, P)$. <u>Want</u> a finite state acceptor (S, A, i, t, F). Immediately, we see the alphabet A is the same and $i =$<s>. This gives us the idea of associating to every nonterminal symbol in $V \backslash A$ a state. <s>$\in V \backslash A$, so that's good. Next we ask:

Q: Is it sufficient for $S = V \backslash A$?