Problems and Comments on Modeling Computation

Rosen, Fifth Edition: Chapter 11; Sixth Edition: Chapter 12

Finite-State Machines with No Output

Section 11.3, Problems: 1, 2, 3, 4, 5, 7, 8, 9, 12,13, 14, 29 (fifth edition); Section 12.3, Problems: 1, 2, 3, 4, 5, 7, 8, 9, 12,13, 14, 29 (sixth edition)

The notations in the Rosen book are non-standard. I follow more the Sipser book which I highly recommend for additional reading.

Let Σ be any **finite** set. Σ will be called the alphabet. **Finite** strings with elements from Σ are called the **words** over Σ. The empty string ε is also considered as a word. Σ^* is the set of all words over Σ. For example, let $\Sigma = \{0,1\}$. Then $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 011, 101, 110, 111, \dots\}$ is the set of all **bitstrings**.

The number of elements in a word over Σ is called its **length** and denoted as $l(w)$. If Σ has n elements, then there are n^m many words of length m. For example, there are $8 = 2^3$ of bitstrings of length 3.

If w and w' are words, then the string ww' is called the product or **concatenation** $w * w'$ of w and w'. We have that this product operation on Σ^* is associative, that is $(w * w') * w'' = w * (w' * w'')$, and clearly $\varepsilon w = w\varepsilon = w$. That is, the mathematical structure $\mathbf{\Sigma} = (\Sigma, *, \varepsilon)$ is a semi group (associative algebra) with unit.

Let A and B be subsets of Σ^*. Then the concatenation of A and B is the set of all products ww' where $w \in A$ and $w' \in B$. We have that $A\varnothing = \varnothing A = \varnothing$ while $A\{\varepsilon\} = \{\varepsilon\}A = A$.

We define $A^0 = \{\varepsilon\}, A^{n+1} = AA^n$. Thus $A^1 = AA^0 = A\{\varepsilon\} = A$. We define

$$A^* = \bigcup_{n \geq 0} A^n = \{w_0 w_1 \cdots w_{n-1} \mid w \in A\}$$

For $A = \Sigma$, this definition is consistent with the previous one. Any subset of Σ^* is called a **language** over the alphabet Σ.

Let Σ be the Latin alphabet and E be the set of words of an English dictionary. Then E is a subset of the English language. Not all English words might be covered.

Let $\Sigma = \{0, 1, 2, \dots, 9\}$. Then every element of $N = \Sigma^* \backslash \{\varepsilon\}$ can be interpreted as a natural number in decimal notation, ignoring leading zeroes. Certain subsets of N are easy to recognize, like the set of numbers that are divisible by 5. Other subsets are much more difficult to recognize, like the set P of those numbers that occur as a segment after the decimal point in the expansion of $\pi = 3.1459\ldots P = \{1, 14, 145, 1459, \dots\}$

We are going to describe those sets which are recognizable by certain finite devices, the so called **finite automata**. The description of a finite automaton or finite state machine has five ingredients:

1. A finite non-empty set S of states.
2. A finite non-empty set Σ of input symbols. It is also called the alphabet.
3. A transition function δ : On the input $\alpha \in \Sigma$, a state $s \in S$ is transformed into to a state $s' \in S$, that is $\partial : S \times \Sigma \to S$, $\partial(s, \alpha) = s'$.
4. An initial or start state s_0.
5. A subset F of S of final or accept states.

Thus $\mathbf{M}$ may be perceived a a five tuple $\mathbf{M} = (S, \Sigma, \delta, s_0, F)$. The transition function is given by a table and a set F:

∂	α_0	$\cdots$	α_j	$\cdots$	α_{n-1}
s_0	$\delta(s_0, \alpha_0)$	$\cdots$	$\delta(s_0, \alpha_j)$	$\cdots$	$\delta(s_0, \alpha_{n-1})$
$\vdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_i	$\delta(s_i, \alpha_0)$	$\cdots$	$\delta(s_i, \alpha_j)$	$\cdots$	$\delta(s_i, \alpha_{n-1})$
$\vdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_{m-1}	$\delta(s_{m-1}, \alpha_0)$	$\cdots$	$\delta(s_{m-1}, \alpha_0)$	$\cdots\ldots\ldots$	$\delta(s_{m-1}, \alpha_{n-1})$

$F = \{\ldots\}$

The central concept of this section is the one of a **computation**. Let w be a word over Σ, that is an element of $\Sigma^*, w = w_1 w_2 \cdots w_k$. Then w is **accepted** by $\mathbf{M}$ if $r_k \in F$ where the sequence $r_0, r_1, \ldots, r_k$ is defined by $r_0 = s_0$, the initial state, and where $r_1 = \delta(s_0, w_1), r_2 = \delta(r_1, w_2), \ldots, r_{i+1} = \delta(r_i, w_{i+1}), \ldots, r_k = \delta(r_{k-1}, w_k)$:

$$s_0 = r_0 \overset{w_1}{\to} r_1 \overset{w_2}{\to} r_2 \to \ldots r_{k-1} \overset{w_k}{\to} r_k$$

w_1 puts the initial state into state r_1, then r_1 is put into state r_2 by signal w_2, and finally w_k puts r_{k-1} into state $r_k \in F$.

For any automaton $\mathbf{M}$ we then have the language recognized by $\mathbf{M}$:

$$L(\mathbf{M}) = \{w \mid w \text{ is accepted by } \mathbf{M}\}$$

Definition *A language $A \subseteq \Sigma^*$ is **regular** if there is some finite automaton such that $A = L(\mathbf{M})$.*

For any alphabet Σ, $\varnothing$ and Σ^* are regular. Just choose any automaton $\mathbf{M}_0$ where $F = \varnothing$ in order to have no word accepted; and an automaton $\mathbf{M}_1$ where $F = S$ in order to have every word accepted.

Exercise *Find an automaton for $\{\alpha\}, \alpha \in \Sigma$.*

The empty string ε is accepted in case that the initial state s_0 is a final state. That makes sense, the state s of the machine $\mathbf{M}$ should not change, if there is no input signal. Thus ε is accepted if the the machine is in an acceptance state.

We can even think that Σ is augmented by ε to $\Sigma \cup \{\varepsilon\} = \Sigma_\varepsilon$. Then $\delta(s, \varepsilon) = s$ holds for every s.

What we have defined as an automaton is also called a **deterministic** automaton. For every $s \in S$ and every $\alpha \in \Sigma_e$, $d(s, \alpha) = s'$ is defined for a unique state s'.

For a **non-deterministic** automaton the transition function $\delta(s,\alpha)$ is a set of states, that is a subset of S, hence an element of the power set $\mathbf{P}(S)$ of S. Of course, we allow $\delta(s,\alpha)$ to be the empty set. We also include ε as an input signal and for a non-deterministic automaton one always has that $\delta(s,\varepsilon) \supseteq \{s\}$. If $s' \in \delta(s,\varepsilon)$ then we say that s' has been obtained from s by an ε –move.

As before, we need five ingredients for the formal definition of a non-deterministic automaton:

1. A finite non-empty set S of states.
2. A finite non-empty set Σ of input symbols. It is also called the alphabet.
3. A transition function $\delta : S \times \Sigma_\varepsilon \to \mathbf{P}(S), \delta(s,\varepsilon) \supseteq \{s\}$.
4. An initial or start state s_0.
5. A subset F of S of final or accept states.

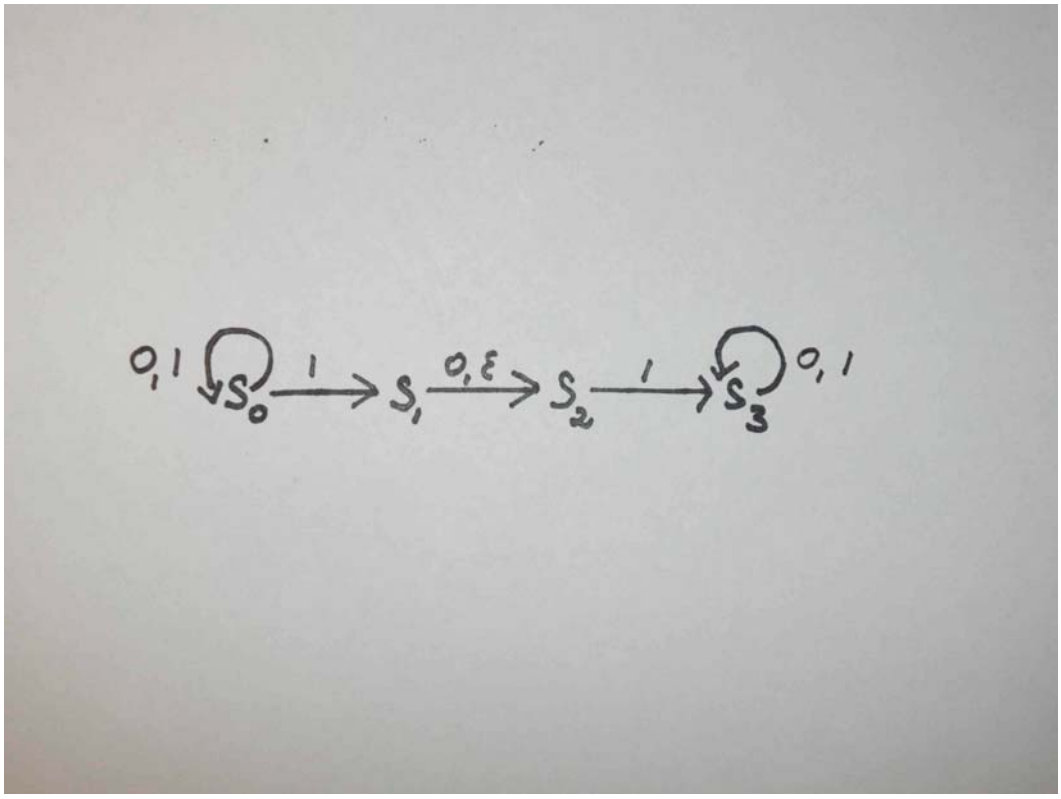
A finite non-determinisic automaton $ND(\mathbf{M})$ accepts a word $w = w_1 w_2 \cdots w_k \in \Sigma^*$ if $w = v_1 v_2 \cdots v_l$ and if there is some sequence $r_0, r_1, \ldots, r_l$ of states in S such that

1. $r_0 = s_0$
2. $r_{i+1} \in \delta(r_i, v_{i+1})$
3. $r_l \in F$

That $w = w_1 w_2 \cdots w_k = v_1 v_2 \cdots v_l$ means that the strings $w_1 w_2 \cdots w_k$ and $v_1 v_2 \cdots v_l$ differ only by insertions of a number of ε moves.

The definition of the language recognized by a non-deterministic automaton is as before

Here is the graph of a non-deterministic automaton:



The non-deterministic automaton $DF(\mathbf{M}_1)$, with final set $F = \{s_3)\}$

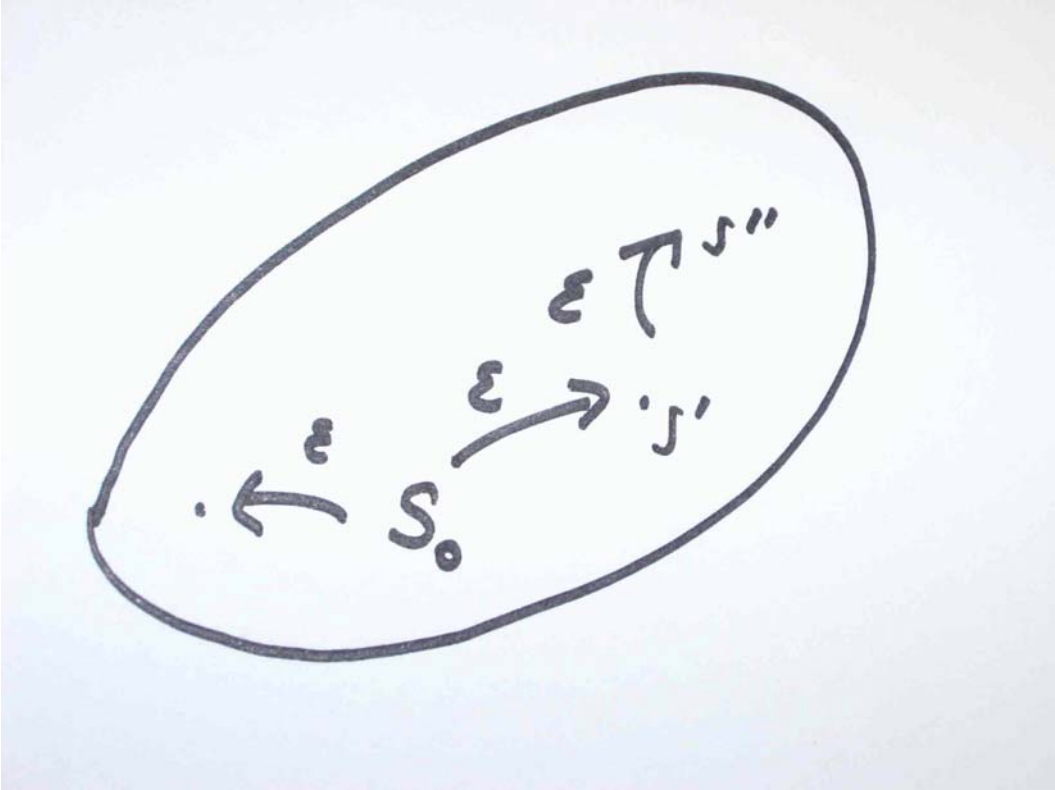
The initial state is s_0. The word $w = 11 = 1\varepsilon 1$ is accepted through the calculation $r_0 = s_0, r_1 = s_1 \in \delta(s_0, 1) = \{s_0, s_1\}, r_2 = s_2 \in \delta(s_1, \varepsilon), r_3 = s_3 \in \delta(s_2, 1)$

In order to understand the process of a non-deterministic calculation better, we need the following

Definition *For any set R of states let*

$$E(R) = \{t \mid t \text{ can be reached from some } s \in R \text{ by zero or more } \varepsilon - \text{moves}\}$$

$E(\{s_0\})$ is the set of all states which can serve as initial state. An ε-move may change s_0 to s' another one s' to s''etc. :



A picture of $E(\{s_0\})$

For a nondeterministic calculation of $w = w_1 w_2 \cdots w_k$ the role of s_0 may be played by any of the elements of $E(\{s_0\})$. The first input w_1 is equal to $\varepsilon\varepsilon\cdots\varepsilon w_1$ and thus is processed at some $s \in E(\{s_0\})$. It is important that s can be any element of $E(\{s_0\})$. We then can get into a state $r \in \delta(s, w_1)$. Because $w_1 = w_1\varepsilon\varepsilon\cdots\varepsilon$, the next state r can be any element of $E(\delta(s, w_1))$. The states r which we can get from input w_1 are elements of the sets $E(\delta(s, w_1))$ where $s \in E(\{s_0\})$. If one of the sets $E(\delta(s, w_1))$ contains an element of F then the word $w = w_1$ got accepted by the non-determinist automaton. Because then there is a path leading from s_0 to some $s \in E(\{s_0\}$ through some ε –moves, then via w_1 to some $t \in E(\delta(s, w_1))$ and then via ε –moves to some $f \in F$.

This process can be thought of that w_1 causes a transition from the initial set $E(\{s_0\})$ to the set $\bigcup\{ E(\delta(s, w_1)) | s \in E(\{s_0\})\}$.

We are now ready to define the deterministic automaton $D(\mathbf{M})$ for the non-deterministic automaton $ND(\mathbf{M})$:

1. The set of states of $D(\mathbf{M})$ is the powerset $\mathbf{P}(S)$ of the set S. That is the states of $D(\mathbf{M})$ are subsets R of the set S of states for $D(M)$. The empty subset $\varnothing$ of S is of course included.

2. The transition function for $D(\mathbf{M})$ is

$$\delta'(R, \alpha) = \bigcup\{E(\delta(r, \alpha)) | r \in R\}$$

where, of course, δ is the transition function for $ND(\mathbf{M})$.

3. The initial set for $D(\mathbf{M})$ is $E(\{s_0\})$
4. The set of final states consists of all those sets R which contain an element of F.

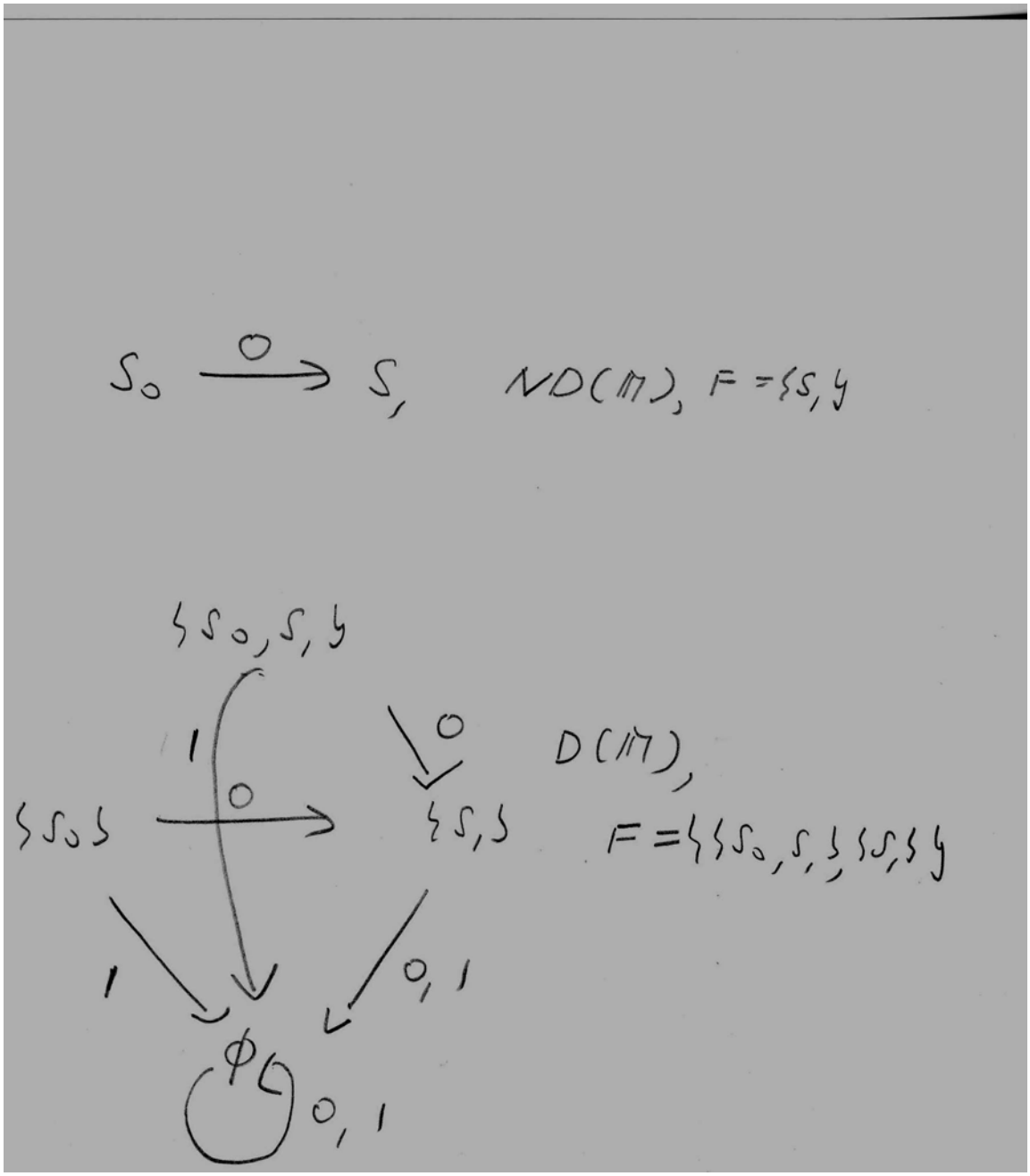
The following is now quite evident:

Theorem *The deterministic automaton* $D(\mathbf{M})$ *for the non-deterministic automaton* $ND(\mathbf{M})$ *recognizes exactly the same words as* $D(\mathbf{M})$.

Let $\Sigma = \{\alpha_1, \ldots, \alpha_n\}$ be an alphabet. Then every singleton $\{\alpha_i\}$ is regular. A non-deterministic automaton $ND(\mathbf{D})$ with two states, s_0 and s_1, and transition function that under α sends the initial state s_0 to the final state s_1 does the job. Let for example $\Sigma = \{0, 1\}$, $\alpha = 0$. Then the deterministic automaton has four states: $\varnothing, \{s_0\}, \{s_1\}, \{s_0, s_1\}$. The initial state is $\{s_0\}$ and there are now two final states $\{s_1\}$ and $\{s_0, s_1\}$. The transition function is given by

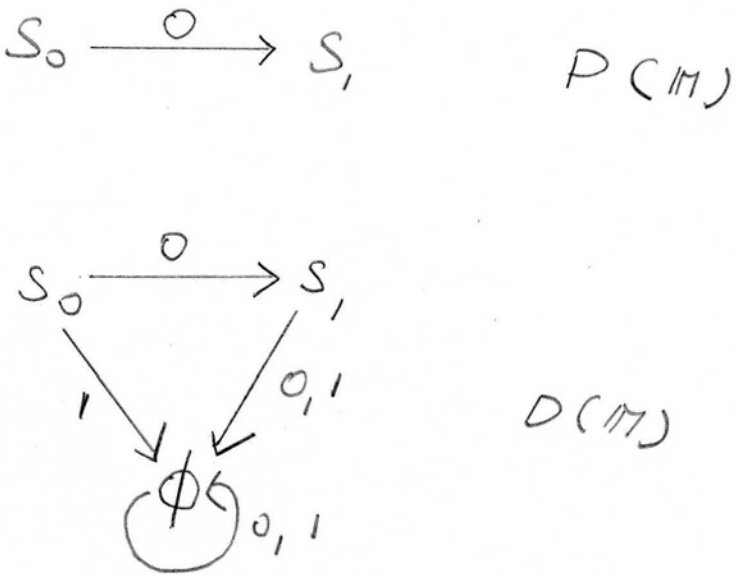
δ	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\{s_o\}$	$\{s_1\}$	$\varnothing$
$\{s_1\}$	$\varnothing$	$\varnothing$
$\{s_0, s_1\}$	$\{s_1\}$	$\varnothing$

Here are the non-deterministic and deterministic automata $ND(\mathbf{M})$ and $D(\mathbf{M})$ that recognize $\{0\}$:



Automata for $\{0\}$

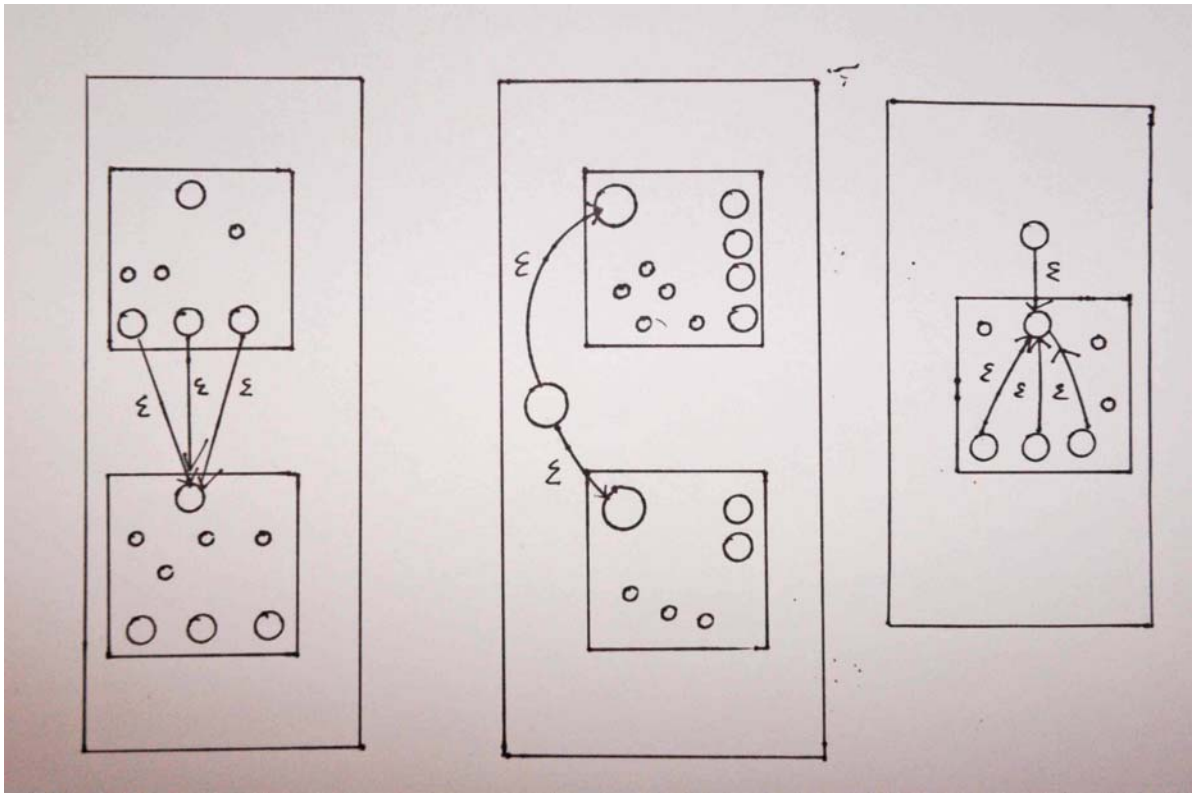
The non-deterministic automaton $ND(\mathbf{M})$ for $\{0\}$ is what is called a **partial automaton**. A partial automaton is like a deterministic one, but its transition function is only partially defined. A partial automaton can be made deterministic by adding one additional non-final state, sometime called failure state $\emptyset$. In case that $\delta(s,\alpha)$ is not defined, define $\delta(s,\alpha) = \emptyset$. Of course, $\delta(\emptyset,\alpha) = \emptyset$, for every $\alpha \in \Sigma$:



Deterministic automaton for $\{0\}$ from $P(\mathbf{M})$

If the set A is regular then $A^c = \Sigma^* \backslash A = \{w \mid w \in \Sigma^*, w \notin A\}$ is regular. For the proof, we choose a deterministic automaton for A and change its set F of final states to $S \backslash A$. If we start with a nondeterministic automaton for A, then we need to find its deterministic equivalent first. The partial automaton for $\{0\}$ has only one final state, s_1, and the complement is $S \backslash \{s_1\} = \{s_0\}$. The partial automaton $P(\mathbf{M})$ with $\{s_0\}$ as set of final states would recognize only ε.

The complement of $A = \{0\}$ in $\{0,1\}^*$ is $\{\varepsilon, 1w, 0w'\}$ where w is any word, ε included, and w' is any word of positive length. In the automaton $D(\mathbf{M})$ for $\{0\}$ we change the set $F = \{s_1\}$ to $S \backslash F = \{s_0, \varnothing\}$. Having the initial state s_0 also as a final state, makes ε accepted. The computation $s_0 \overset{1}{\to} \varnothing$ recognizes the word $w = 1$, and the words $1w$ are recognized by the computations $s_0 \overset{1}{\to} \varnothing \overset{w}{\to} \varnothing$; the computations $s_0 \overset{0}{\to} s_1 \overset{0,1}{\to} \varnothing \overset{w}{\to} \varnothing$ recognize all words of length at least two which start with 0.



Machines for concatenation, union and star

If a deterministic machine M recognizes the language A and a deterministic machine N recognizes B then we can easily define a nondeterministic L machine which recognizes $A \circ B$. In the picture we indicate the start and final states by larger circles, start at the top and final states at the bottom. The machine L has as its set of states the union of states of M and N. Of course, we may assume that M and N have no states in common. For every final state of M there is an ε –move to the start state of N. If a word w is accepted by L, then $w = w_1 \varepsilon w_2$ where w_1 is accepted by M, that is input w_1 ends at a final state of M and then an ε –move is yields the start state of N and processing of w_2 must yield to a final state because w is accepted by L. That is, the language for L is contained in $A \circ B$. On the other hand if $w = w_1 \circ w_2$ where $w_1 \in A$ and $w_2 \in B$ then $w = w_1 w_2 = w_1 \varepsilon w_2$ shows that w is recognized by L.

A non-deterministic machine L that recognizes $A \cup B$ is easily constructed by taking a new start state and ε –moves that connect this new state with he initial states of machines for A and B. A word w then is accepted if it is accepted by one of the machines for A or B.

The non-deterministic machine L for A^* needs a new start state which is also a final state to make sure that ε is accepted. This new start state is connected by an ε –move with the start state of the machine N for A. Every final state of N is connected by an ε –move with the start state for N. Clearly a word w is recognized by L either it is ε or if $w = w_1 \varepsilon w_2 \varepsilon \cdots \varepsilon w_k$ where $w_i \in A$. That is $w \in A^*$.

Theorem *The set* $\mathbf{R}(\Sigma^*)$ *of regular languages over* Σ *contains* $\varnothing, \Sigma^*, \{\varepsilon\}, \{\alpha\}, \alpha \in \Sigma$, *and is closed under taking complements, finite unions, concatenations and the star*

operation.

Corollary $\mathbf{R}(\Sigma^*)$ *is closed under finite intersections. That is* $\mathbf{R}(\Sigma^*)$ *is a boolean algebra of subsets of the powerset algebra* $\mathbf{P}(\Sigma^*)$.

Proof *Closure under finite intersections follows from DeMorgan laws:* $A \cap B = (A^c \cup B^c)^c$.

.

Are there languages that are not recognized by any automaton? That is, are there languages that are not regular? Indeed, there are non-regular languages.

The simplest example of such a language is the set of all words 0^n1^n over the alphabet $\Sigma = \{0, 1\}$. The proof is quite simple. Assume that there is a deterministic automaton $\mathbf{M}$ which recognizes the language $L = \{w | w = 0^n1^n\}$. We are going to show that $\mathbf{M}$ necessarily recognizes words that are **not** of the form 0^n1^n. The proof is based on a *pumping argument*.

Let k be the number of states of $\mathbf{M}$ and let $w = 0^{2k}1^{2k}$. Because w is recognized by $\mathbf{M}$, we have a computation

$$s_0 = r_0 \overset{0}{\rightarrow} r_1 \overset{0}{\rightarrow} r_2 \rightarrow \ldots r_{2k-1} \overset{0}{\rightarrow} r_{2k} \overset{1}{\rightarrow} r_{2k+1} \overset{1}{\rightarrow} \ldots \overset{1}{\rightarrow} r_{4k}$$

where s_0 is the initial state an…d where r_{4k} is a final state. But amongst the first $2k$ states for computing the head 0^{2k} of $w = 0^{2k}1^{2k}$ there must be a repetition, say in

$$s_0 = r_0 \overset{0}{\rightarrow} r_1 \overset{0}{\rightarrow} r_2 \rightarrow \ldots r_{2k-1} \overset{0}{\rightarrow} r_{2k}$$

we have that $r_j = r_{j+s}$. But then $\mathbf{M}$ recognizes $0^{2k+s}1^{2k}$ where we may think that s –many zeroes have been inserted after the first j –zeroes.

Language Recognition

Section 11.4, Problems 1, 2, 3, 4, 7, 8 (a,b,c) Sectiom 12.4 Problems 1, 4, 7, 8, 13, 14 (a,b,c) (sixth edition)

Let Σ be an alphabet. We consider $\varnothing$, and ε just as auxiliary "symbols" that don't belong to Σ. The set of *regular* expressions is defined recursively:

1. $\varnothing$ is regular.
2. Each $\alpha \in \Sigma$ is regular; ε is regular.
3. Assume that $\mathbf{S}$ and $\mathbf{T}$ are regular expressions. Then

$$(\mathbf{S} \vee \mathbf{T}), (\mathbf{ST}), \mathbf{S}^*$$

are regular.

A string $\mathbf{R}$ is regular if it is obtained by finitely many applications of the rules 1-3.

Example $((\alpha \vee \beta)^*\gamma), (\varepsilon\alpha), \alpha^*, (\alpha \vee \varnothing)$ *are regular. If* $\Sigma = \{\alpha, \beta, \gamma\}$ *are the elements of* Σ, *then* $\mathbf{R} = ((\alpha \vee \beta) \vee \gamma))$ *and* $\mathbf{R}^* = ((\alpha \vee \beta) \vee \gamma))$ *are regular.*

Let $REG(\Sigma)$ denote the set of regular expressions. We define recursively a map L from

the set $REG(\Sigma)$ of regular expressions into the set $\mathbf{R}(\Sigma)$ of regular sets:

$$L \ : \ REG(\Sigma) \to \mathbf{R}(\Sigma), \mathbf{R} \mapsto L(\mathbf{R})$$

$$\emptyset \mapsto \emptyset, \varepsilon \mapsto \{\varepsilon\}, \alpha \mapsto \{\alpha\},$$

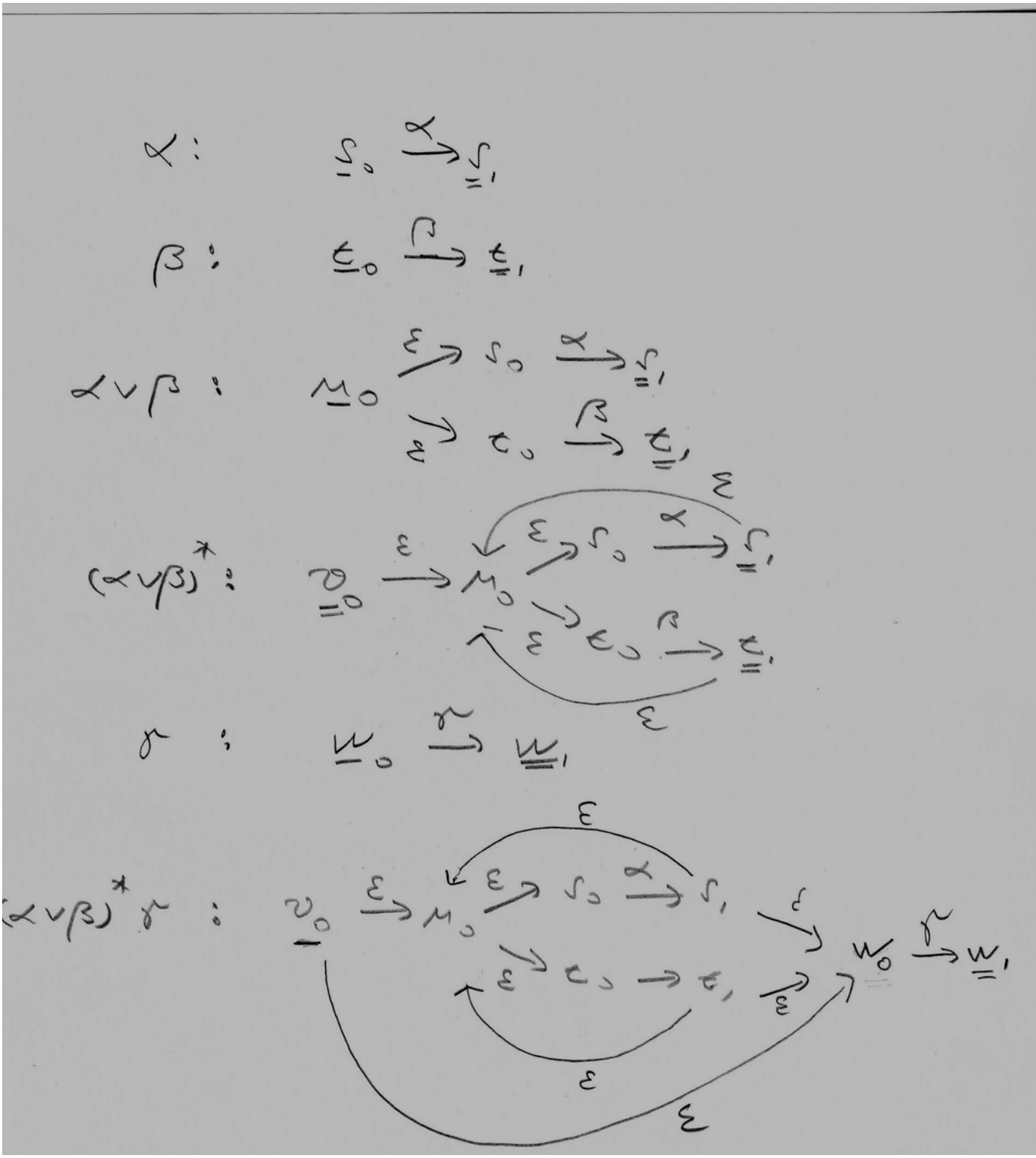
$$L(\mathbf{S} \vee \mathbf{T}) \ = \ L(\mathbf{S}) \cup L(\mathbf{T}), L(\mathbf{ST}) = L(\mathbf{S}) \circ L(\mathbf{T}), L(\mathbf{S}^*) = L(\mathbf{S})^*$$

The empty set $\emptyset$, and the singletons $\{\alpha\}$ and $\{\varepsilon\}$ are regular. Because the union of regular sets is regular, $L(\mathbf{S} \vee \mathbf{T})$ is regular for regular expressions $\mathbf{S}$ and $\mathbf{T}$. A similar argument applies for concatenation and the star operation to see that L is a map from the regualr expressions into the set of regular sets.

Example $L((\alpha \vee \beta)^* \gamma)) \ = L(\alpha \vee \beta)^* \circ L(\gamma) = (L(\alpha) \cup L(\beta))^* \circ L(\gamma) = (\{\alpha\} \cup \{\beta\})^* \circ$ *of all words that end in* γ.

From a regular expression $\mathbf{R}$ we can in a systematic fashion construct a non deterministic automaton for the regular set $L(\mathbf{R})$.

Example *Find a non-determinist automaton for the set of all words in* $\Sigma = \{\alpha, \beta, \gamma\}$ *that end in* γ, *that is for* $L((\alpha \vee \beta)^* \gamma))$.



Machine for $R_1 = (\alpha \vee \beta)^* \circ \gamma$

Initial states have been underlined once, final states underlined twice. Leftmost states

are always initial.

By abuse of language, we may identify regular sets A, B with their recognizing non-deterministic automata.

Recall the basic rules:

In order to construct $A \cup B$, we introduce a new initial state s and connect s by means of ε –moves with the initial states u_0 of A and v_0 of B, respectivley. The set of final states of $A \cup B$ is the union of the final states of A and B.

In order to construct $A \circ B$ we connect all final states of A with the initial state v_0 of B. The final states of B are the final states of B.

For A^* we need a new initial state s which is also a final state of A^*. By doing this is, we have recognized ε. The new initial state s is connected by an ε –move with the initial state of A. The final states of A^* are the final states of A.

Exercise *Find a non-deterministic automaton for all bit strings of even length. We first have to find a regular expression* $\mathbf{R}$ *for this set. Now, any such string can be divided into bit strings of length* 2, *that is into strings* $00, 01, 10, 11$. *Hence* $R = (00 \vee 01 \vee 10 \vee 11)^*$. *Then apply the basic rules for the construction of a* $ND(\mathbf{R})$.

We have seen that for every regular expression $\mathbf{R}$ we can find an automaton that recognizes the set $L(\mathbf{R})$. However, the converse is also true. Given any regular set A then there is some regular expression $\mathbf{R}$ such that $L(\mathbf{R}) = A$. Both statements are the content of Kleene's theorem:

Theorem *Regular expressions correspond to regular sets. That is, every regular expression* $\mathbf{R}$ *describes a unique regular set* $L(\mathbf{R})$, *and for every regular set* A *there is some regular expression* $\mathbf{R}$ *such that* $L(\mathbf{R}) = A$.

Proof Let A be any regular set. It is recognized by some deterministic or non-deterministic automaton $\mathbf{M}$. We somehow must manage to find a procedure to read off from any non-determinist automaton $\mathbf{M}$ the regular expression it recognizes. In some cases this is easy, like in Figure 1:

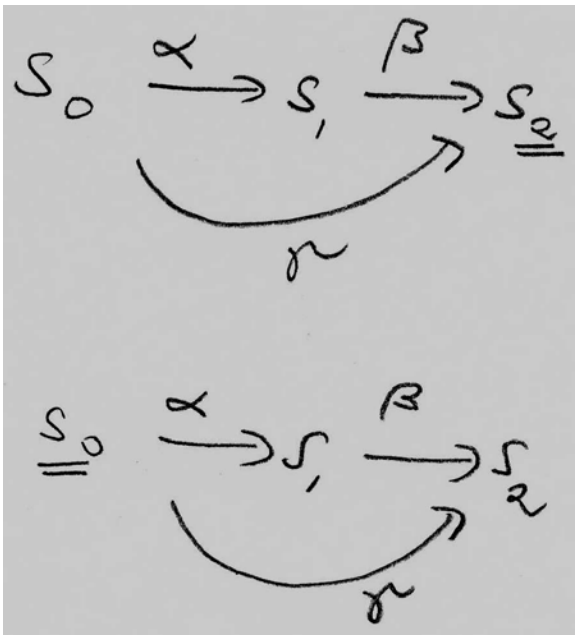
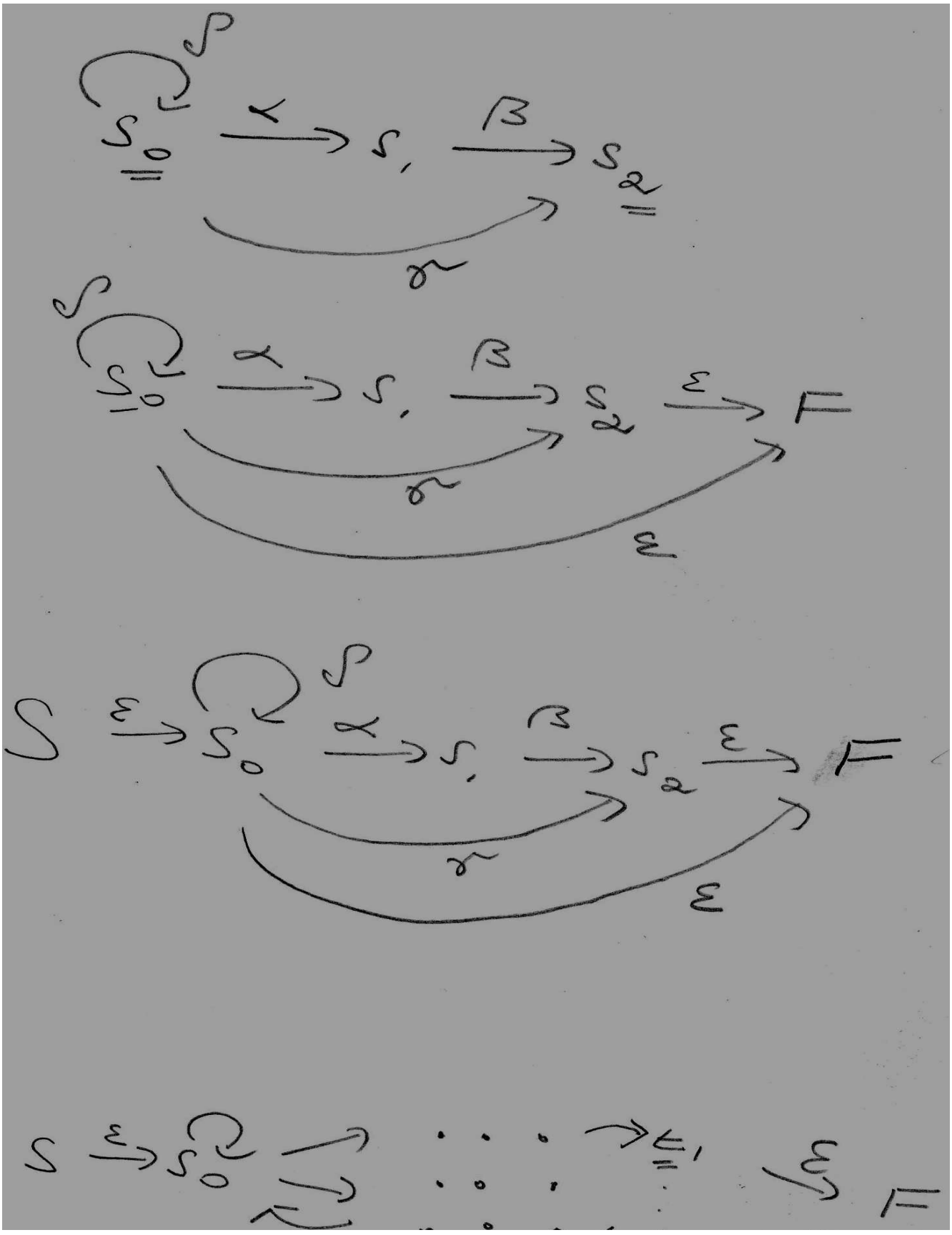


Figure 1

Clearly, the machine on top recognizes the set with regular expression $a\beta \vee \gamma$ while the other machine recognizes in addition to this set also ε, that is,

it recognizes $\varepsilon \vee a\beta \vee \gamma$. By abuse of language, we have identified an expression with its regular set. Slightly more complicated is the automaton on top of Figure 2:


S
S0
α
S1
β
S2
α
S
S0
α
S1
β
S2
ε
F
ε
S
ε
S
S0
α
S1
β
S2
ε
F
α
ε
S
ε
S0
F
ε

It recognizes the set $\varepsilon \vee \delta^* a\beta \vee \gamma$.

By adding a new final state, *F*, and ε –moves from the old fianl states, s_2 and s_0 we have gotten an automaton with exactly one final state. Similarly, we may add a new initial state, *S*, and an ε –move to s_0. By doing this we have arrived at an automaton where there is no incoming arrow to the start state *S*, and no outgoing arrow from the unique final state *F*. This can be achieved for any non-deterministic automaton without changing its accept set.

If we have several arrows $\alpha_1, \ldots, \alpha_n$ going from a state *s* to a different state *t* then we may combine them to one single arrow, but labelled by the regular expression $\alpha_1 \vee \ldots \alpha_n$. Similarly for loops about a state *s*. If there is no arrow from *s* to *t* then we can introduce one and label it by 0. If there is no loop about *s* then we can add one and label it by ε. See Figure 3:

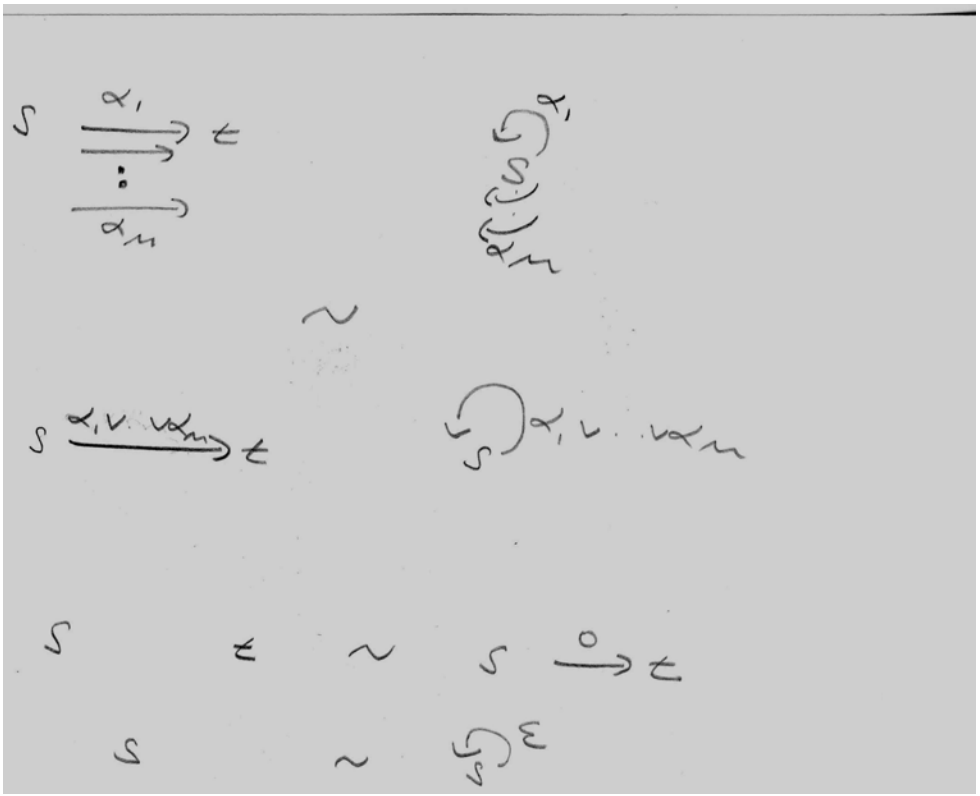


Figure 3

It is now cllear that we can convert any non-deterministic automaton into one which has exactly one final state *F* and where there is no incoming arrow into the start state *S* and no outgoing arrow from *F*. For every pair of states *s* and *t*, where $s \neq F$ and $t \neq S$ there is exactly one aroow labelled by some regular expression.

We can formally define a *non-deterministic* ***generalized*** *finite automaton, GND*(**M**) as a five tuple:

$$\mathbf{M} = (S, \Sigma, \delta, q_s, q_a), \text{where}$$
$$\delta : (Q \backslash \{q_a\}) \times (Q \backslash \{q_s\}) \rightarrow REG(\Sigma), \delta(q_i, q_j) = \mathbf{R}$$

for which we write $q_i \overset{\mathbf{R}}{\rightarrow} q_j, q_i \neq q_a, q_j \neq q_s$

The machine **M accepts** $w = w_1 \dots w_k, w_i \in \Sigma^*$, if there is a sequence of states $q_0, q_1, \dots q_k$ such that $q_0 = q_s =$ **start state**, $q_k = q_a =$ **accept state** and for each $i = 1, \dots k$ one has that $w_i \in L(\mathbf{R}_i), R_i = \delta(q_{i-1}, q_i)$.

In case that **M** has only two states, that is $Q = \{q_s = S, q_a = F\}$, $R = \delta(S, F), S \overset{\mathbf{R}}{\rightarrow} T$, we have that a word w is accepted if and only if $w \in L(\mathbf{R})$.

Here is an example of a computation for the generalized automaton in Figure 4:

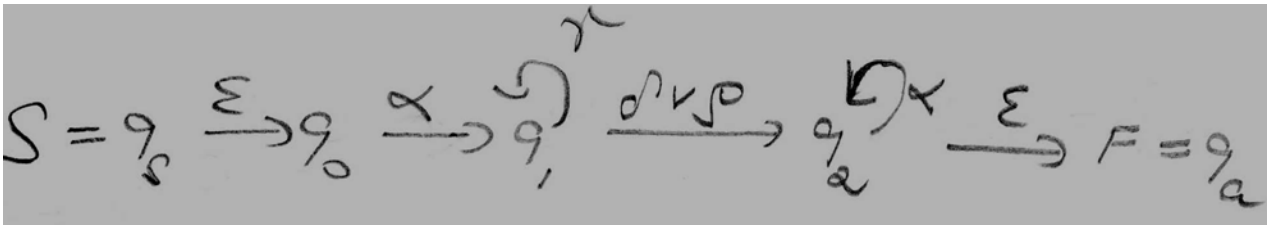


Figure 4

$w = \varepsilon\alpha\gamma\gamma\delta\alpha\varepsilon$ is computed along states $q_s q_0 q_1 q_1 q_1 q_2 q_2 q_a$. We have that $\varepsilon \in L(\varepsilon) = \{\varepsilon\}, \varepsilon = \delta(q_s, q_0); \alpha \in L(\alpha), \alpha = \delta(q_0, q_1); \gamma \in L(\gamma), \gamma = \delta(q_1, q_1);$ $\gamma \in L(\gamma), \gamma = \delta(q_1, q_1); \delta \in L(\delta \vee \rho), \delta \vee \rho = \delta(q_1, q_2), \alpha \in \delta(q_2, q_2), \varepsilon \in L(\varepsilon), \varepsilon =$

The proof of Kleene's Theorem will be finished if we can show that every generalized automaton can be converted into one with only two states. This can be done because we can convert any generalized automaton with more than two states into an equivalent one which has one state less. Let **M** be a generalized automaton with k states where $k > 2$. we pick any state q which is different from q_s and q_a. we call this state q_{rip} because we "rip" it out. In order to understand how it works, recall that for a generalized automaton there is exactly one arrow $q_i \overset{R}{\rightarrow} q_j$ as long as $q_i \neq q_a$ and $q_j \neq q_s$. In Figure 5, we show how the uique arrow with label R_4 which goes from q_i to q_j hjas to be replaced becasause of ripping q_{rip}, using the unique arrows to q_{rip}, about q_{rip}, and from q_{rip}.

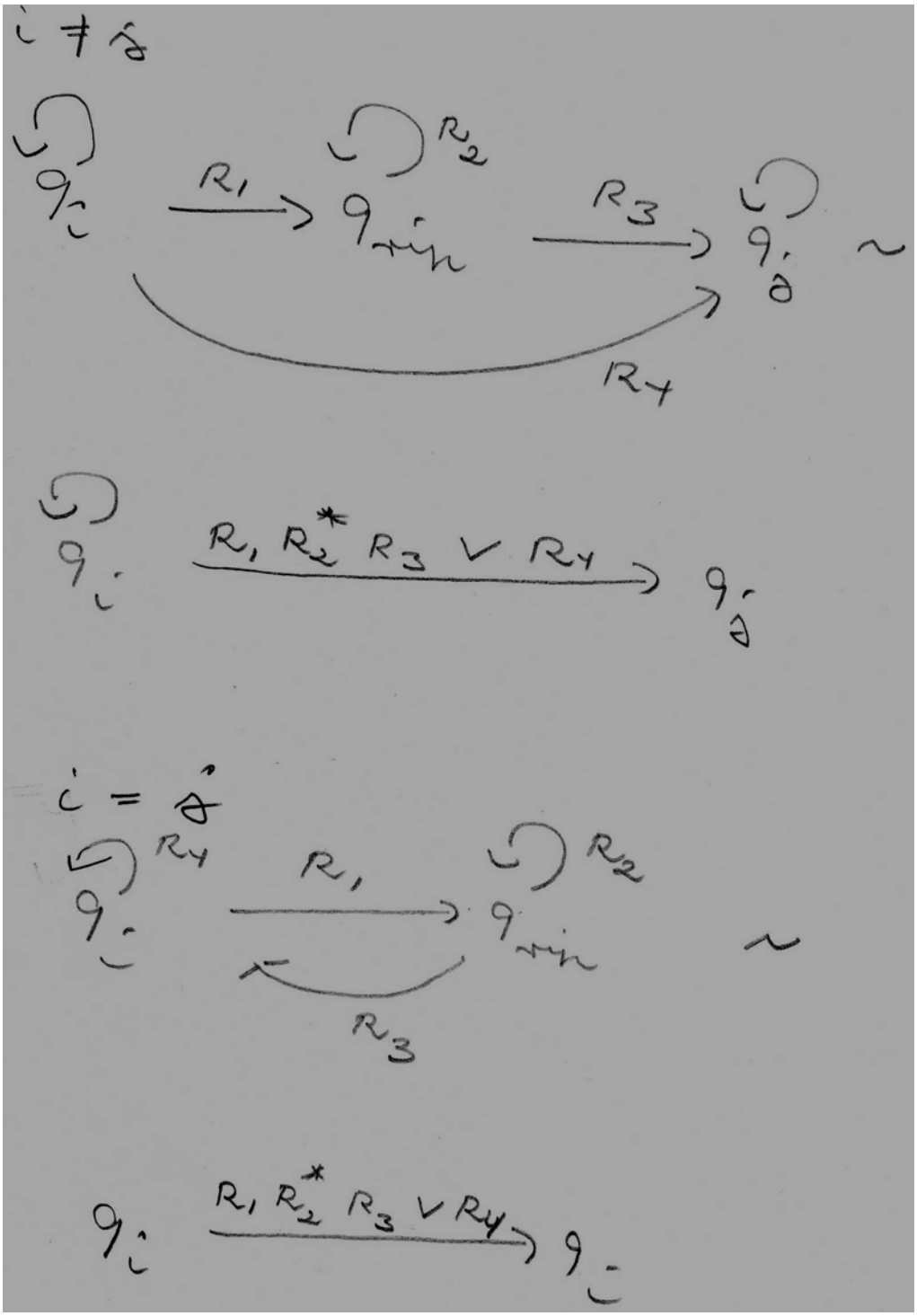


Figure 5

We can continue this process until we get an automaton with two states and the regular expression **R** for the regular set *A*. This concludes the proof.

Figure 6 illustrates this process

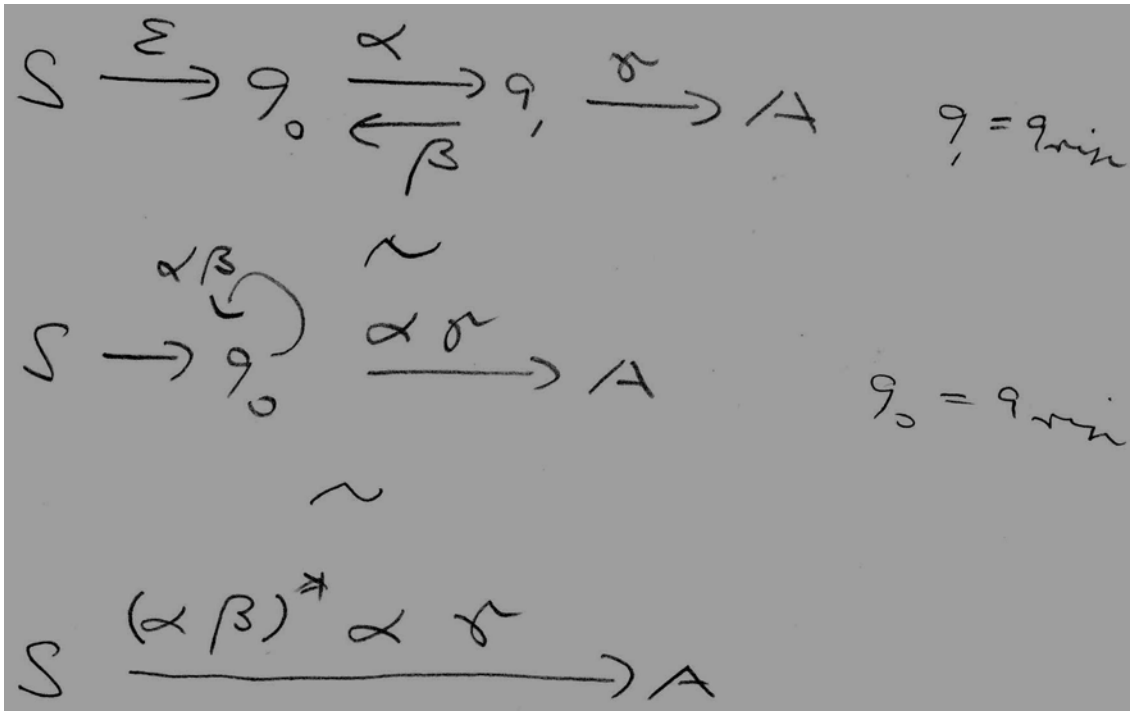


Figure 6

In Figure 7 we have a somewhat more complicated example:

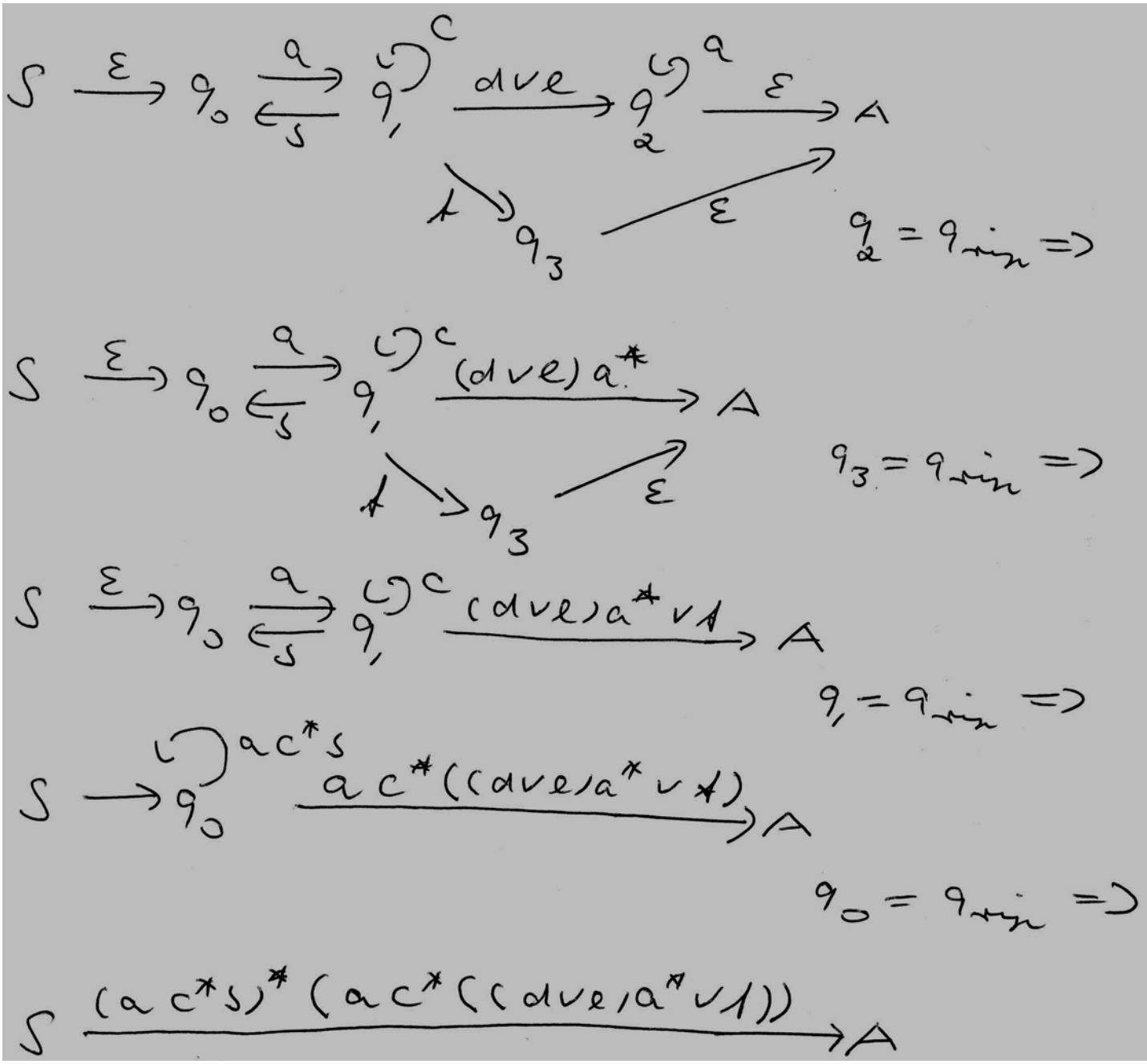


Figure 7

Let A be a regular set. We know that its complement A^c is also regular. In order to find the regular expression for A^c, say starting with a non-deterministic automaton for $\mathbf{N}$ for A, one first has to find a deterministic automaton $\mathbf{M}$ from $\mathbf{N}$. By taking the complement of its acceptance set, one gets a deterministic automaton for A^c. Then, as before, one constructs a generalized automaton from which one gets a regular expression $\mathbf{R}^c$ for A^c, which is of course an expression in $\circ, \vee,$ and *. There is no rule to get from the regular expression $\mathbf{R}$ an expression $\mathbf{R}^c$ such that $L(\mathbf{R}^c) = A^c$. A similar remark applies to the regular expression, call it $(\mathbf{R} \wedge \mathbf{S})$, for the intersection of the regular sets

$A = L(\mathbf{R})$ and $B = L(\mathbf{S})$. There is no general formula for $(\mathbf{R} \wedge \mathbf{S})$, using $\mathbf{R}$ and $\mathbf{S}$ and the operations $\circ, \vee,$ and * of the Kleene algebra. We only know that for every choice of R and S there is such an expression.

Section 11.1, Problems 1, 2, 3, 5, 6, 14, 15, 16, 17, Sectiom 12.1 Problems 1, 2, 3, 7, 8, 20, 21, 22, 23 ,(sixth edition)

In the previous sections we have learned that regular languages can be recognized by finite state machines and compactly described by elements of a certain boolean algebra, the Kleene algebra.

Regular languages can also be described by a generation process which involves an alphabet and *"production rules"*. In order to do this one starts with the general concept of a *phrase-structure grammar* G which defines a language $L(G)$. Restrictions on the production rules define then the important classes of *"context-free"* languages and the *"regular languages".* As a main theorem we will prove **Chomsky's Theorem** which characterizes regular languages as those that are generated by *"regular grammars".*

A phrase-structure grammar is a four tuple $G = (V, T, S, P)$ where

1. V is a finite, non-empty set*;*
2. T is a non-empty subset of V of *terminals;*
3. S is a element of $V \backslash T$, the *start symbol ;*
4. P is a finite set of *productions.*

Now, what are productions? As usual, V^* is the set of all words over the alphabet V. A production then is a set of ordered pairs (w, w'). That is

$$P \subseteq V^* \times V^*$$

Instead of $(z, z') \in P$ one says that $z \to z'$ is a production of G. By definition, z and z' are words over V.

The elements of $V \backslash T$ may be thought of as *variables* and usually denoted by capital Latin letters, like $A, B, C, \ldots,$ while terminals are denoted by lower case letters, like $a, b, c, \ldots.$

In $z \to z'$, the right-hand side may be the empty word λ. However, we assume that the left-hand side z is never the empty string.

Let G be a given phrase-structure grammar, $z \to z'$ be a production of G and w be a word over V, which has z as a substring. That is, $w = lzr$. Here l and r stand for left and right, respectively. Then we say that the word $w' = lz'r$ is *directly derivable* from w and write $w \Rightarrow w'$.

Example $V = \{S, 0, 1\}; \{S \to 0S1\ , S \to \lambda\} = P.$ *Then* $0S1 \Rightarrow 00S11$ *is a direct derivation. Here* $T = \{0, 1\}$ *is the set of ter*mina*ls.*

For a sequence of direct derivations $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots, w_{n-1} \Rightarrow w_n$ we write $w_0 \overset{*}{\Rightarrow} w_n,$ and say that w is *derivable* from w_0.

Example *Continuing the last example,we have that*
$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 0000S1111 \Rightarrow 00001111$ *is a derivation of*

0000111 *from S.*

Let G be a phase-structure grammar. The set of words, which contain only terminals and which are derivable from S , is called the *language* $L(G)$ *generated by* G :

$$L(G) = \{w \in T^* | \, S \overset{*}{\Rightarrow} w\}$$

Example *For* $V = \{S, 0, 1\}; \{S \to 0S1 \, , S \to \lambda\} = P$ *we obviously have that* $L(G) = \{0^n 1^n | \, n \in \mathbb{N}\}$. *We have shown before that* $L(G)$ *is not recognizable by any finite state automaton.*

Example *Using again the same* $V = \{S, 0, 1\}$ *but a different set of productions,* $S \to \lambda$, $S \to 0S, S \to S1$, *we see that* $S \Rightarrow 0S \Rightarrow 0S1$. *If we now use* $S \to 0S$, *then we add a* 0 *to the left of S, if we use* $S \to S1$, *then we add a* 1 *to the right of* S : $S \Rightarrow 0S \Rightarrow 0S1 \Rightarrow 00S1 \Rightarrow 000S1 \Rightarrow 000S11 \Rightarrow 00011$. *It is quite obvious that this grammer generates the regular language* $L(G) = \{0^n 1^m | \, n, m \in \mathbb{N}\}$

A phrase-structure grammar is called *context-free* if all productions are of the form $A \to w$, where A is a variable (non-terminal) symbol of V. That is an application of such a rule replaces an occurrence of a variable A in a string σ by w, where w is a string of elements of V.

A grammar is called *context-sensitive* if the rules are of the form $lAr \to lwr$. That is, if a variable A occurs in a string σ and if it is surrounded by the string l from the left and by the string r from the right, then A can be replaced by w.

Recall that the productions of an arbitrary phrase-structure grammar are of the form $z \to z'$ which means that a substring z in a word σ can be replaced by z'. If theree are no restrictions on productions, then the phrase-structure grammar is called of *type 0.*

Context-sensitive grammars are also called of *type 1.* A variable A can be replaced by w only within context l and r.

Context-free grammars are called of *type 2.* The rules of a context-free grammar allow replacement of A in a string σ by the string w.

Remark *Replacement of a variable A in a string* σ *by w is different from what we do in mathematics when we substitute a variable x by w. For example, in* $x + (x \sin x)$, *a substitution of x by* π *yields* $\pi + \pi \sin \pi$. *Each occurrence of x in the expression* $\sigma = x + (x \sin x)$ *must be replaced by* π. *An application of a production* $A \to w$ *in a string s means that some occurences (but not neccessarily all) of A in* σ *have been replaced by w.*

Context free languages can be used for the description of formal expressions, like terms in algebra. We wish to formalize that sums and products of terms are terms. This suggests rules:

$$S \to (S + S), S \to (S \cdot S), S \to a|b|c|$$

The vocabulary of our grammar for producing terms contains only one variable, S, and terminals $T = \{(,), +, \cdot, 0, a, b, c\}$. Thus $V = S \cup T$, Of course, (stands for the left paranthesis ,) for the right paranthesis, $+$ is the addition symbol and $\cdot$ the multiplication symbol. The terminals a, b, c are thought of as "variables" ranges over an algebraic domain.

We use shorthand $A \to w_1|w_2 \ldots$ for $A \to w_1, A \to w_2, \ldots$

Example $S \Rightarrow (S+S) \Rightarrow ((S \cdot S)+S) \Rightarrow ((S \cdot S)+(S+S)) \Rightarrow ((a \cdot S)+(S+S)) \Rightarrow \ldots$ *shows that* $((a \cdot b)+(a+c)) \in L(G)$. *The language* $L(G)$ *consists of all algebraic expressions involving addition and multiplication in* a, b, c.

In order to describe algebraic expression in any number of variables for a particular structure, like the natural numbers $\mathbb{N}$ with addition and multiplication, $\mathbb{N} = (N, +, \cdot)$, we need to generate infinitely many variables and all natural numbers out of a finite vocabulary. This is not much of a problem. In addition to rules generating terms, we need rules to generate variables and numerals

$$S \to N, N \to 0N, N \to \lambda, S \to U, U \to xN, .$$

The idea is that a sequence of n –many zeroes stands for the numeral n, and a followed by m –many zeroes stands for the variable x_m.

Exercise 1 Define explicitly a phrase-structure grammar whose language $L(G)$ is the set of all terms in variables x_m and natural numbers. In particular derive $((x_3 \cdot 2)+(3+1))$.

Theorem *Let* $L = L(G), L_1 = L(G_1), L_2 = L(G_2)$ *be languages that are generated by context-free phrase-structure grammars. Then* $L^*, L_1 \circ L_2$ *and* $L_1 \cup L_2$ *are context free.*

This is an assigned exercise (15, 21 respectively).

In order to prove the exercise, one adds a new start symbol S_0 and one new rule r_0.

Here is what you have to do for L^* :The grammar G^* for L^* is

$G^* = (V \cup \{S_0\}, T, S_0, P \cup \{r_0\})$ where $r_0 = S_0 \to S_0 S | \lambda$.

Recall that L^* consists of any number of concatenations of words in L. That is, if $S \overset{*}{\Rightarrow} w_1, S \overset{*}{\Rightarrow} w_2, \ldots S \overset{*}{\Rightarrow} w_n$, then $w_1 w_2 \cdots w_n \in L^*$. Now, n –many applications of rule r_0 yield $S_0 \Rightarrow S_0 S \Rightarrow S_0 S S \Rightarrow \cdots S_0 S S \cdots S \Rightarrow S S \cdots S$. And then using the derivation in G we get $S_0 \overset{*}{\Rightarrow} w_1 w_2 \cdots w_n$.

The other cases are quite similar. Of course, we may assume that the vocabularies V_1 and V_2 for G_1 and G_2 are disjoint.

A phrase structure grammar is called a *regular grammar* if all rules are of the form $A \to aB$ or $A \to a$ where A is a non-terminal symbol and a is a terminal symbol. If S is the start symbol, also $S \to \lambda$ is allowed.

Exercise 2 Show that the rules of a regular grammars can be generalized to $A \to wB$, $A \to w$, $S \to \lambda$ where w is a non-empty word over T

Regular grammars generate regular languages and vice versa. This is an important theorem The key is to associate to the transition function of an automaton $\mathbf{M}$ rules of a regular grammar. Thus let

$$\mathbf{M} = (S, \Sigma, \delta, s_0, F)$$

be a finite state machine. In order to get a clue what G should be we want to associate to a transition $\delta(a, s) = s'$ a rule. The natural choice is the rule $s \to as'$. Of course that

makes sense only if s and s' are non-teminal symbols and a a terminal symbol. If s' is a final state then the corresponding rule should be $s \to a$. Thus we define as set V of our grammar G the union $V = S \cup \Sigma$ where the set T of terminal symbols is $T = \Sigma$. The start symbol of G should be the start state s_0 of $\mathbf{M}$.

As an illustration how this works, assume that the word abc is recognized by the machine $\mathbf{M}$, that is we have a sequence of states:

$$s_0 \overset{a}{\to} s_1 \overset{b}{\to} s_2 \overset{c}{\to} s_3 \in F$$

This translates into a derivation

$$s_0 \Rightarrow as_1 \Rightarrow abs_2 \Rightarrow abc$$

This is fine as long s_3 is not s_0. In case that s_0 is a final state we have that the empty word is accepted, and we must add the rule $s_0 \to \lambda$ in order to have $L(\mathbf{M}) = L(G)$.

Thus, in case that $\delta(a,s) = s_0 \in F$ we define rules for G, $s \to as_0,\ s_0 \to \lambda$.

For our example, in case that $s_3 = s_0 \in F$ we get $s_0 \Rightarrow as_1 \Rightarrow abs_2 \Rightarrow abcs_0$ and we either can finish with $s_0 \Rightarrow as_1 \Rightarrow abs_2 \Rightarrow abc$, using the rule $s_0 \to \lambda$ or continue $s_0 \Rightarrow as_1 \Rightarrow abs_2 \Rightarrow abcs_0 \Rightarrow abcas_1 \Rightarrow abcabs_2 \Rightarrow abcabcs_0$ etc.

It is quite obvious that $L(\mathbf{M}) = L(G)$.

Now assume that we are given a regular grammar $G = (V,T,S,P)$. We wish to define a machine $\mathbf{M} = (S,\Sigma,\delta,s_0,F)$ such that $L(\mathbf{M}) = L(G)$. The start state s_0 of $\mathbf{M}$ should be the start symbol of G. That is:

$$S \text{ of } G = s_0 \text{ of } \mathbf{M}$$

The set V of G consist of variables A and the set T of terminal symbols. We take as the set S of **non-final** states for M the subset of non-terminal symbols of V. That is

$$V \backslash T \text{ of } G = S \backslash F \text{ of } \mathbf{M}$$

The set T of terminals a of G is the input alphabet Σ of $\mathbf{M}$. That is

$$T \text{ of } G = \Sigma \text{ of } \mathbf{M}$$

Now, how do production rules translate into the definition of δ? First, our machine will be non-deterministic. For a rule $A \to aB$ we define $\delta(a,A) = B$. Of course we also may have a rule $A \to aB$. So, $\delta(a,A) = B$ is allowed. Thus, $\mathbf{M}$ will be non-deterministic. That is,

$$\text{If } A \to aB \text{ is a production of } G \text{ then } \delta(a,A) = B \text{ is a transition for } \mathbf{M}$$

Now, what should we do about a production $A \to a$? It is quite natural to say that $\delta(a,A)$ should be a final state. It cannot be a variable. So we add λ to the set of states.

$$\text{If } A \to a \text{ is a production of } G \text{ then } \delta(a,A) = \lambda \text{ is a transition for } \mathbf{M}$$

In case that $S \to \lambda$ is a production of G then we have to add S as a final state:

The set of final states for $\mathbf{M}$ is just $\{\lambda\}$ and $\{\lambda, s_0 = S\}$ in case that $S \to \lambda$ is a production of G

Let for example be
$S = S_0 \Rightarrow aS_1 \Rightarrow abS_2 \Rightarrow abcS_0 \Rightarrow abcaS_1 \Rightarrow abcabS_2 \Rightarrow abcabcS_0 \Rightarrow abcabc$ be a chain of productions where we used $S_0 \to aS_1, S_1 \to bS_2, S_2 \to c, S_2 \to cS_0, S_0 \to \lambda$. This yields for $\mathbf{M}$ the sequence of state transitions

$S_0 \overset{a}{\Rightarrow} S_1 \overset{b}{\Rightarrow} S_2 \overset{c}{\Rightarrow} S_0 \overset{a}{\Rightarrow} S_1 \overset{b}{\Rightarrow} S_2 \overset{c}{\Rightarrow} S_0 \overset{\lambda}{\Rightarrow} \lambda$

Thus we have demonstrated the following theorem of Chomsky:

Theorem *A set is regular if and only if it is generated by a regular grammar.*