D1. Alphabet, string, empty string, length of string, substring, concatenation, reversal, language, Kleene-star (closure).

- **Alphabet** is a finite set of symbols.
- A **string** over an alphabet is a finite sequence of symbols from the alphabet.
- A string may have no symbols at all; in this case it is called the empty string and is denoted by *e*.
- The length of a string *w* is its length as a sequence, denoted by $|w|$.
- Two strings over the same alphabet can be combined to form a third by the operation of concatenation. The concatenation of strings *x* and *y*, written *x*o*y* or simply *xy*, is the string *x* followed by the string *y*. Formally:
  $w = x\text{o}y \Leftrightarrow \quad |w| = |x| + |y|$
  $w(j) = x(j) \qquad \text{for } j = 1,..., |x|$
  $w(|x| + j) = y(j) \qquad \text{for } j = 1,..., |y|$
- A string *v* is a substring of *w* iff there are strings *x* and *y* such that *w* = *xvy*.
- The reversal of a string *w*, denoted by w^R is the string "spelling backwards".
  1. If *w* is a string of length 0, then $w^R = w = e$.
  2. If *w* is a string of length $n+1>0$, then *w* = ua for some $a \in \Sigma$, and $w^R = au^R$

- Any set of string over an alphabet Σ - that is any subset of Σ^*- is called a **language**.
- Closure or Kleene star is a language operation of a single language L, denoted by L^*. L^* is the set of all strings obtained by concatenating zero or more strings from L.

D2. Regular Expression over an alphabet; Language represented by a regular expression.
The **regular expression** over an alphabet Σ is a string over the alphabet $\Sigma \cup \{(,), \emptyset, \cup, *\}$ such that the following hold.

(1) Ø and each member of Σ is a regular expression
(2) If α and β are regular expressions then so is (αβ)
(3) If α and β are regular expressions then so is (α∪β)
(4) If α is a regular expression, then so is α^*.

Nothing is a regular expression unless it follows from (1) through (4)

Every regular expression represents a language. Formally, the relation between regular expressions and the languages they represent is established by a function L, such that if α is any regular expression, then L(α) is the language represented by α. Thus, L is a function from strings to languages. The function L is defined as follows.

(1) L(Ø) = Ø and L(α) = {a} for each $\alpha = a \in \Sigma$
(2) If α and β are regular expressions, then L((αβ))= L(α)L(β).
(3) If α and β are regular expressions, then L((α∪β))= L(α)∪L(β).
(4) If α is a regular expression then $L(\alpha^*) = L(\alpha)^*$.

D3. Deterministic finite automaton (K, Σ, δ, s, F), configuration, relation yields; languages accepted by deterministic finite automata.

Deterministic finite automaton is a quintuple M = (K, Σ, δ, s, F)
where
K is the set of states (finite),
Σ is an alphabet,
$s \in K$ is the initial state,
F is the set of final states ($F \subseteq K$),
and
δ the transition function, is a function from K×Σ to K.

A configuration is determined by the current state and the unread part of the string being processed. In other words, a configuration of a deterministic finite automaton (K, Σ, δ, s, F) is any element of K×Σ*.
If (*q*, *w*) and (*q*', *w*') are two configurations of M, then (*q*, *w*) $\vdash_M$ (*q*', *w*') iff *w* = σ*w*' for some symbol $\sigma \in \Sigma$, and δ(*q*, σ) = *q*'. In this case we say that (*q*, *w*) yields (*q*', *w*') in one step. (The binary relation $\vdash_M$ holds between two configurations of M iff the machine can pass from one to the other as a result of a single move.) We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$; (*q*, *w*) $\vdash_M^*$ (*q*', *w*') is read, (*q*, *w*) yields (*q*', *w*').
A string $w \in \Sigma^*$ is said to be accepted by M iff there is a state $q \in F$ such that (s, *w*) $\vdash_M^*$ (*q*, *e*). The language accepted by M, L(M), is the set of all strings accepted by M.

D4. Nondeterministic finite automaton (K, Σ, Δ, s, F), configuration, relation yields; languages accepted by nondeterministic finite automata.
Non-deterministic finite automaton is a quintuple M = (K, Σ, Δ, s, F)
where
K is the set of states (finite),
Σ is an alphabet,
s ∈ K is the initial state,
F is the set of final states (F ⊆ K),
and
Δ the transition relation, is a finite subset of K×Σ*×K.

A configuration of M is an element of K×Σ*.
The relation $\vdash_M$ between configuration (yields in one step) is defined as follows: $(q, w) \vdash_M (q', w')$ iff there is a $u \in \Sigma^*$ such that $w = uw'$ and $(q, u, q') \in \Delta$. $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$.
$w \in \Sigma^*$ is accepted by M iff there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, e)$.
L(M), the language accepted by M, is the set of all strings accepted by M.

D5. Equivalence of finite automata. Theorem on the equivalence (no proof).
Finite automata M_1, and M_2 are said to be equivalent iff $L(M_1) = L(M_2)$.
For each non-deterministic finite automaton, there is an equivalent deterministic finite automaton.

D6. Context-free grammar (V, Σ, R, S); context-free languages.
A **context-free grammar** G is a quadruple (V, Σ, R, S), where
V is an alphabet,
Σ (the set of terminals) is a subset of V,
R (the set of rules) is a finite subset of (V-Σ)×V*, and
S (the start symbol) is an element of V-Σ.

The members of V-Σ are called nonterminals. For any $A \in$ V-Σ and $u \in V^*$, we write $A \rightarrow_G u$ wherever $(A, u) \in$ R. For any strings $u, v \in V^*$, we write $u \Rightarrow_G v$ iff there are strings $x, y, v' \in V^*$ and $A \in$ V-Σ such that $u = xAy$, $v = xv'y$ and $A \rightarrow_G v'$. The relation $\Rightarrow_G^*$ is the reflexive, transitive closure of $\Rightarrow_G$.

L(G), the language generated by G, is $\{w \in \Sigma^*: S \Rightarrow_G^* w\}$; we also say that G generates each string in L(G). A language L is said to be a context-free language if it is equal to L(G) for some context-free grammar G.

D7. Regular grammars.
A context-free grammar G = (V, Σ, R, S) is regular iff $R \subseteq (V-\Sigma)\times\Sigma^*((V-\Sigma)\cup e)$.

D8. Pushdown automaton (K, Σ, Γ, Δ, s, F); transition, push, pop, configuration, relation "yields", language accepted.
A **pushdown automaton** is a sixtuple M = (K, Σ, Γ, Δ, s, F), where
K is a finite set of states,
Σ is an alphabet (the input symbols),
Γ is an alphabet (the stack symbols),
s is the initial state,
F is the set of final states, and
Δ is the transition relation, is a finite subset of $(K\times\Sigma^*\times\Gamma^*)\times(K\times\Gamma^*)$.

Intuitively, if $((p, u, \beta), (q, \sigma)) \in \Delta$, then M, whenever it is in state p with β at the top of the stock, may read u from the input tape, replaced by σ on the top of the stack, and enter state q. Such a pair $((p, u, \beta), (q, \sigma))$ is called a transition of M.

To push a symbol is to add it to the top of the stack, to pop a symbol is to remove it form the top of the stack.

A configuration is defined to be a member of $K\times\Sigma^*\times\Gamma^*$: The first component is the state of the machine, the second is the partition of the input yet to be read, and the third is the content of the pushdown store, read top-down.

For every transition $((p, u, \beta),(q, \sigma)) \in \Delta$, and for every $x \in \Sigma^*$ and $\alpha \in \Gamma^*$, we define $(p, ux, \beta\alpha) \vdash_M (q, x, \sigma\alpha)$), moreover $\vdash_M$ (yields in one step) holds only between configurations that can be represented in this form for some transition, some x, and some α.

We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$, and say that M accepts a string $w \in \Sigma^*$ iff $(s, w, e) \vdash_M^* (p, e, e)$ for some $p \in F$. The language accepted by M, denoted L(M) is the set of all strings accepted by M.

D9. Turing machine (K, Σ, δ, s); configuration, relation "yields"; computation.
A **Turing machine** is a quadruple (K, Σ, δ, s) where

K: is a finite set of states, not containing the halt state h; ($h \notin K$),
σ: is an alphabet, containing the blank symbol #, but not containing the symbols L and R;
s: is the initial state;
δ: is a function from $K \times \Sigma$ to $(K \cup \{h\}) \times (\Sigma \cup \{L, R\})$.

Configuration of a Turing machine M = (K, Σ, δ,s) is a member of
$(K \cup \{h\}) \times \Sigma^* \times \Sigma \times (\Sigma^*(\Sigma - \{\#\}) \cup \{e\})$.

Let M = (K, Σ, δ, s) be a Turing machine and let (q_1, w_1, a_1, u_1) and (q_2, w_2, a_2, u_2) be configurations of M. Then
$(q_1, w_1, a_1, u_1) \vdash_M (q_2, w_2, a_2, u_2)$

iff for some $b \in \Sigma \cup \{L, R\}$, $\delta(q_1, a_1) = (q_2, b)$ and either
(1) $b \in \Sigma$, $w_1 = w_2$, $u_1 = u_2$, and $a_2 = b$,
or
(2) $b = L$, $w_1 = w_2 a_2$, and either
  (a) $u_2 = a_1 u_1$, if ($a_1 \neq \#$ or $u_1 \neq e$),
  or
  (b) $u_2 = e$, if $a_1 = \#$ and $u_1 = e$.
or
(3) $b = R$, $w_2 = w_1 a_1$ and either
  (a) $u_1 = a_2 u_2$
  or
  (b) $u_1 = u_2 = e$ and $a_2 = \#$.

For any Turing machine M, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$;
We say configuration C_1 yields configuration C_2 if $C_1 \vdash_M^* C_2$.

A computation by M is a sequence of configurations $C_0, C_1, C_2, \ldots, C_n$ for some $n \geq 0$ such that
$C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.

D10. Turing computable function; Turing acceptable languages; Turing decidable languages.
Let f be a function from Σ_0^* to Σ_1^*. Turing machine M = (K, Σ, δ, s) is said to compute f if $\Sigma_0, \Sigma_1 \subseteq \Sigma$ and for $w \in \Sigma_0^*$, if $f(w) = u$, then

$$(s, \#w\underline{\#}) \vdash_M^* (h, \#u\underline{\#}).$$

If such Turing machine exists, the function is called **Turing computable**.

Let Σ_0 an alphabet not containing the blank symbol. Let Ⓨ and Ⓝ be two fixed symbols not in Σ_0. Then the language $L \subseteq \Sigma_0^*$ is **Turing decidable** iff function $x_L: \Sigma_0^* \rightarrow \{Ⓨ, Ⓝ\}$ is Turing computable, where for each $w \in \Sigma_0^*$,

$$x_L(w) = \begin{cases} Ⓨ \text{ if } w \in L \\ Ⓝ \text{ if } w \notin L \end{cases}$$

Let Σ_0 an alphabet not containing #. M accepts a string $w \in \Sigma_0^*$ if M halts on input w. Thus M accepts language $L \subseteq \Sigma_0^*$ iff, $L = \{w \in \Sigma_0^*: M \text{ accepts } w\}$, and a language is **Turing acceptable** if there is some Turing machine that accepts it.

D11. Machine schema (μ, η, M_0). Turing machine (K, Σ, δ, s) representing machine schema (μ, η, M_0).
A **machine schema** is a triplet (m, η, M_0) where

m is a finite set of Turing machines with a common alphabet Σ and disjoint state sets;
$M_0 \in$ m is the initial machine; and
η is a function from subset of $m \times \Sigma$ to m.

Let m = $\{M_0, M_1, \ldots, M_m\}$ ($m \geq 0$), where $M_i = (K_i, \Sigma, \delta_i, s_i)$ for $i = 0, 1, \ldots, m$. Let $q_0, q_1, \ldots, q_m$ be new states not in any of the K_i. Then machine schema (m, η, M_0) represents the Turing machine M, where M = (K, Σ, δ, s)

$K = K_0 \cup \ldots \cup K_m \cup \{q_0, q_2, \ldots, q_m\}$,
$s = s_0$, and

and δ is defined is follows.

(a) if $q \in K_i (0 \le i \le m)$, $a \in \Sigma$, $\delta_i(q, a) = (p, b)$, and $p \neq h$, then
$\delta(q, a) = \delta_i(q, a) = (p, b)$;
(b) if $q \in K_i (0 \le i \le m)$, $a \in \Sigma$, and $\delta_i(q, a) = (h, b)$ then $\delta(q, a) = (q_i, b)$;
(c) if $a \in \Sigma$ and $\eta(M_i, a)$ $(0 \le i \le m)$ is not defined, then $\delta(q, a) = (h, a)$;
(d) if $a \in \Sigma$ and $\eta(M_i, a) = M_j (0 \le i \le m)$ and $\delta_j(s_j, a) = (p, b)$, then

$$\delta(q_i, a) = \begin{cases} (p, b) \text{ if } p \neq h \\ (q_j, b) \text{ if } p = h \end{cases}$$

D12. Universal Turing Machines (major ideas)
The only feature possessed by electronic computers but not by Turing machines is programmability.

Difficulties:
- The alphabet of any Turing machine is fixed and finite.
- The set of states of any Turing machine is also fixed and finite.

Ideas:
- Assume that there are fixed countably infinite sets

$$K_\infty = \{q_1, q_2, q_3, \ldots\}$$

and

$$\Sigma_\infty = \{a_1, a_2, a_3, \ldots\}$$

such that for every Turing machine, the set of states is a finite subset of K_∞ and the alphabet is a finite subset of Σ_∞.

- Encode the alphabet and states of the Turing machine representing the program.

σ	$\lambda(\sigma)$
q_i	I^{i+1}
h	I
L	I
R	II
a_i	I^{i+2}

- Encode the Turing machine M over the alphabet $\{c, I\}$.

$$p(M) = cS_0cS_{11}S_{12} \ldots S_{1l}S_{21}S_{22} \ldots S_{k1} \ldots S_{kl}c$$

S_0 encodes the initial state, $S_0 = \lambda(s)$.
S_{pr} encodes the values of the transition function, $S_{pr} = cw_1cw_2cw_3cw_4$, where

$$w_1 = \lambda(q_{i\,p}),\ w_2 = \lambda(q_{j\,r}),\ w_3 = \lambda(q'),\ w_4 = \lambda(b)$$

for each

$$\delta(q_{i\,p}, q_{j\,r}) = (q', b).$$

- The symbols on the tape must also be in encoded form.

$$p(w) = c\lambda(b_1)c\lambda(b_2)c \ldots c\lambda(b_n)c$$

for each

$$w = b_1b_2 \ldots b_n.$$

- Universal Turing machine U simulating any Turing machine M is a three tape Turing machine.
Tape 1: encoding of the tape of Turing machine M.
Tape 2: encoding of the Turing machine M.
Tape 3: encoding of the current state of the Turing machine M in the simulation

T1. Languages accepted by finite automata are closed under union (proof).
$L(M) = L(M_1) \cup L(M_2)$
K_1 and K_2 are disjoint.
Construct a non-deterministic $M = (K, \Sigma, \Delta, s, F)$
$K = K_1 \cup K_2 \cup \{s\}$
s = new state not in $K_1 \cup K_2$.
$F = F_1 \cup F_2$
$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}$

T2. Languages accepted by finite automata are closed under concatenation (proof).

$L(M) = L(M_1) \circ L(M_2)$
Construct a non-deterministic $M = (K, \Sigma, \Delta, s, F)$

$K = K_1 \cup K_2$
$s = s_1$
$F = F_2$
$\Delta = \Delta_1 \cup \Delta_2 \cup (F_1 \times \{e\} \times \{s_2\})$

T3. Languages accepted by finite automata are closed under Kleen-star (proof).
$L(M) = L(M_1)^*$
Construct a non-deterministic $M = (K, \Sigma, \Delta, s, F)$

$K = K_1 \cup \{s\}$
$s =$ a new state not in K_1
$F = F_1 \cup \{s\}$
$\Delta = \Delta_1 \cup (F \times \{e\} \times \{s\}) \cup \{(s, e, s_1)\}$

T4. Languages accepted by finite automata are closed under complementation (proof).
$L(M) = \Sigma^* - L(M_1)$
Construct a deterministic $M = (K, \Sigma, \delta, s, F)$

$K = K_1$
$s = s_1$
$F = K_1 - F_1$
$\delta = \delta_1$

T5. Languages accepted by finite automata are closed under intersection (proof).
$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$ $\quad (L_1 \cap L_2 = (L_1^c \cup L_2^c)^c)$

T6. There is an algorithm for answering the following question: Given a finite automaton M, is $L(M)=\Sigma^*$? (proof).

- $L(M) ?= \varnothing$?
  Can be answered by checking whether there is any sequence of arrows on the state diagram that leading from the initial state to a finite state; this can be early done since the state diagram is finite.
- $L(M) ?= \Sigma^* \equiv L(M') = \Sigma^* - L(M) ?= \varnothing$

T7. There is an algorithm for answering the following question: Given two finite automata M_1 and M_2, is $L(M_1) \subseteq L(M_2)$? (proof).
$L(M_1) ?\subseteq L(M_2) \equiv (\Sigma^* - L(M_2)) \cap L(M_1) ?= \varnothing$

T8. There is an algorithm for answering the following question: Given two finite automata M_1 and M_2, is $L(M_1)=L(M_2)$? (proof).
$L(M_1) ?= L(M_2) \equiv ?(L(M_1) \subseteq L(M_2)$ and $L(M_2) \subseteq L(M_1))$

T9. Context-free languages are closed under union (proof).
$L(G) = L(G_1) \cup L(G_2)$
$G =$ (V, Σ, R, S)

$V = V_1 \cup V_2 \cup \{S\}$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$
$S =$ a new symbol not in $V_1 \cup V_2$

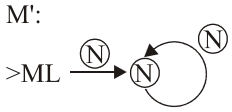
T10. Context-free languages are closed under concatenation (proof).
$L(G) = L(G_1) \circ L(G_2)$
$G =$ (V, Σ, R, S)

$V = V_1 \cup V_2 \cup \{S\}$
$\Sigma = \Sigma_1 \cup \Sigma_2$
$R = R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$
$S =$ a new symbol not in $V_1 \cup V_2$

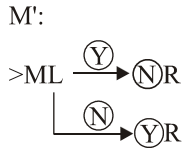
T11. Context-free languages are closed under Kleen-star (proof).
$L(G) = L(G_1)^*$
$G = (V, \Sigma, R, S)$
$V = V_1$
$\Sigma = \Sigma_1$
$R = R_1 \cup \{S_1 \rightarrow e, S_1 \rightarrow S_1S_1\}$
$S = S_1$

T12. Theorems. Turing-decidable languages are Turing acceptable. The complement of a Turing-decidable language is also Turing –decidable.

- Every Turing decidable language is Turing acceptable.



- If L Turing decidable language, then it's complement L^c is also Turing-decidable.



T13. Theorem. A language is Turing-decidable if and only if both it and its complement are Turing acceptable.
→

L is a Turing decidable ⇒ L^c is also Turing decidable
L, L^c is Turing decidable ⇒ L, L^c is Turing acceptable

←

2-tape machine:

M_1 accepts L
M_2 accepts L^c } parallel simulation → which halts ?