Chapter 1 Elements of Language Theory

1.1 Mathematical Preliminaries

Let A and B be two sets. Then

$R \subseteq A \times B$ **relation** *R* **from** *A* **to** *B.*

any subset of the Cartesian product of A and B,

where $A \times B = \{(a, b) | a \in A, b \in B\}$.

A is the ***domain*** *and B the* ***range*** *of R.*

R is a relation ***on*** *A if* $A = B$ *or* $R \subseteq A \times A$.

We write a R b, if $(a, b) \in R$, *a is* ***R-related*** *to b.*

If $A' \subseteq A$, $R(A') = \{b \in B | a\ R\ b$ *for some* $a \in A'\}$

the ***image*** *of A'* ***under*** *R.*

We may write R(a) instead of R({a}).

$R^{-1}: B \times A$ ***inverse*** *of R*

$R^{-1} \equiv \{(b, a) \in B \times A | a\ R\ b\}$

Let $R_1 \subseteq A \times B$ *and* $R_2 \subseteq B \times C$. *Then* $R_1 \cdot R_2 \subseteq A \times C$.

$R_1 \cdot R_2 \equiv \{(a, c) \in A \times C | a\ R_1\ b, b\ R_2\ c, {}^{\exists}b \in B\}$

(relational) ***product*** *of relation* R_1 *and* R_2

or $(R_2 \circ R_1)$

composition *of* R_1 *and* R_2

Fact 1.1 *Multiplication of relations is an* ***associative*** *binary relation.*

$R_1 \cdot (R_2 \cdot R_3) = (R_1 \cdot R_2) \cdot R_3$

Let a relation R is on A.

(1) ***reflexive****, if* $^{\forall} a \in A, a\ R\ a$*; or*

R includes the ***identity relation*** *on A:* id_A.

$id_A = \{(a, a) | \ a \in A\} \subseteq R.$

(2) ***symmetric****, if* $a\ R\ b \Rightarrow b\ R\ a$*; or*

$R = R^{-1}$.

(3) ***antisymmetric****, if* $a\ R\ b$ *and* $b\ R\ a \Rightarrow a = b$*; or*

$R^{-1} \cap R \subseteq id_A$.

(4) ***transitive****, if* $a\ R\ b$ *and* $b\ R\ c \Rightarrow a\ R\ c$*; or*

$R \cdot R \subseteq R$.

The $\mathbf{n^{th}}$ ***power*** *of R(or* ***n-fold product*** *of R)*

$R^0 = id_A$.

$R^n = R \cdot R^{n-1}$, *for* $n > 0$.*(recursive definition)*

Fact 1.2 $a\ R^n\ b$, *if and only if*

$a_0, \ldots, a_n \in A, a = a_0, b = a_n, a_i\ R\ a_{i+1}\ 0 \le i < n.$

*The **transitive closure** of R, denoted by R^+, is*

$R^+ = \cup_{n=1}^{\infty} R^n$.

*The **reflexive transitive closure** of R, R^*, is*

$R^* = \cup_{n=0}^{\infty} R^n$.

***Lemma 1.3** Let R be a relation on a set A.*
*Then R^+ is the **smallest transitive relation** on A that includes R; and*
*R^* is the **smallest reflexive** and **transitive relation** on A that includes R. In other words,*

(1) R^+ is transitive and $R \subseteq R^+$.
(2) $R^+ \subseteq R'$ whenever R' is a transitive relation on A such that $R \subseteq R'$.
(3) R^* is reflexive and transitive and $R \subseteq R^*$.
(4) $R^* \subseteq R'$ whenever R' is a reflexive and transitive relation on A such that $R \subseteq R'$.

Let $A' \subseteq A$.

$R^+(A')$: image of A' under R^+
***positive closure** of A' under R.*
$R^*(A')$: image of A' under R^*
***closure** of A' under R*

A pair G = (A, R) is a ***directed graph(graph)***
A is a set, and R is a relation on A.
$a \in A$ ***node(vertex)*** *of G*
$(a, b) \in R$ ***edge(arc)*** *of G*
edge *(a, b)* ***leave*** *node a*
enter *node b*
a is a ***predecessor*** *of node b*
b is a ***successor*** *of node a*
path of length n *from* a_0 *to* a_n,
a sequence of nodes $(a_0, a_1, \ldots, a_n)$, $n \geq 0$;
if $a_i \; R \; a_{i+1}$, $0 \leq i \leq n$.

$a \; R^n \; b$, *iff* $\exists$ *a* ***path of length n*** *from a to b.*
$a \; R^* \; b$, *iff* $\exists$ *a* ***path*** *from a to b.*
$a \; R^+ \; b$, *iff* $\exists$ *a* ***path of positive length*** *from a to b.*

$G^+ = (A, R^+)$ ***transitive closure*** *of G*
$G^* = (A, R^*)$ ***reflexive transitive closure*** *of G*
$G' = (A', (A' \times A') \cap R)$ ***subgraph*** *of G* $(A' \subseteq A)$

A ***cycle*** *is a path of positive length*
from a node to itself.
A graph is called ***cyclic****, if it contains a cycle,*
acyclic *otherwise.*

*An acyclic graph (A, R) is a **tree**, if*

*(1) $^{\exists}r \in A$, called **root**, such that*

(2) $^{\forall}a \in A$, $^{\exists}$exactly one path from r to a.

(a, b) $\in R$ in a tree *a is the **father** of b*

*b is a **son** of a*

(a, b) $\in R^+$ in a tree *a is an **ancestor** of b*

*b is an **descendent** of a*

*A node having no son is called **leaf**.*

*A **subtree** of (A, R) is*

any tree (A', R') which is a subgraph of (A, R)

no node is an ancestor of any nodes in A \ A'.

*A relation R on a set A is an **equivalence**,*

if it is reflexive, symmetric, and transitive.

$[a]_R \equiv R(a) = \{b | \ a \ R \ b\}$

***equivalent class** of a **under** R*

***Fact 1.5** For an equivalence relation R on A,*

*$\{[a]_R | \ a \in A\}$ is a **partition**.*

***Conversely**, if $\mathbb{P}$ is a partition on a set A,*

$R = \{(a, b) | \ a, b \in B$ for some $B \in \mathbb{P}\}$

is an equivalence relation on A with

$\mathbb{P} = \{[a]_R | \ a \in A\}$.

Fact 1.5 *For an equivalence relation R on A,*
$\{[a]_R \mid a \in A\}$ *is a* ***partition****.*

Conversely*, if* $\mathbb{P}$ *is a partition on a set A,*
$R = \{(a, b) \mid a, b \in B$ *for some* $B \in \mathbb{P}\}$
is an equivalence relation on A with
$\mathbb{P} = \{[a]_R \mid a \in A\}$.

Proof

Let R be an equivalence relation on A.
$\cup_{a \in A} [a]_R = \{b \mid a\ R\ b, a \in A\}$*(reflexive)*
$\supseteq \{a \mid a\ R\ a, a \in A\} = A$
$\cup_{a \in A} [a]_R \subseteq A$ *is trivial.* $\therefore \cup_{a \in A} [a]_R = A$.
Assume that $aR\ b$*,and* $[a]_R \cap [b]_R \neq \emptyset$.
$^{\exists}c$ *.э.* $c \in [a]_R$ *and* $c \in [b]_R$.
$\therefore$ $a\ R\ c$ *and* $b\ R\ c$.
$c\ R\ b$ *(symmetric)*
$a\ R\ b$ *(transitive)* $\therefore$ *contradiction*$(aR\ b)$

Let $\mathbb{P}$ *be a partition on a set A, and*
$R = \{(a, b) \mid a, b \in B$ *for some* $B \in \mathbb{P}\}$.
i) $^{\forall}a \in A, {}^{\exists}B$ *.э.* $a \in B, B \in \mathbb{P}, \therefore a\ R\ a$.
ii) $a, b \in B, B \in \mathbb{P}, a\ R\ b$ *and* $b\ R\ a$.
iii) $a, b, c \in B, B \in \mathbb{P}, a\ R\ b, b\ R\ c$, *and* $a\ R\ c$.
$\mathbb{P} = \{[a]_R \mid a \in A\}$.

*A relation R on A is a (**reflexive**) **partial order**,*
if it is (reflexive), antisymmetric, and transitive.
*A partial order R is a **total order(linear order)**,*
if $^{\forall}a, b \in A$, either a R b or b R a.
*If R is a partial order, the relation $R \setminus R^0$ is a **irreflexive partial order**.*
We often use $\leq$ to denote a partial order,
$<$ to denote a irreflexive partial order $\leq \setminus \leq^0$.
$\therefore$ $a < b$, iff $a \leq b$ and $a \neq b$.
$\leq^{-1} \equiv \geq$, $<^{-1} \equiv >$

*If $\leq$ is a partial order, $(A, \leq)$ **partially ordered set**.*
*If $\leq$ is a total order, $(A, \leq)$ **totally ordered set**.*

*$a \in A$ is **maximal** with respect to a partial order $\leq$*
on A, if $a < b$ is false $^{\forall}b \in A$.
*$a \in A$ is **minimal** with respect to a partial order $\leq$*
on A, if $b < a$ is false $^{\forall}b \in A$.

If $\leq$ is a total order, A can have at most one maximal elements and at most one minimal element.
*When these exist, we call them **maximum** and **minimum** of A respect to $\leq$, denoted by*
$\max_{\leq} A$ and $\min_{\leq} A$ (or max A, min A for short)

$f: A \to B$ *a relation f from A to B*

partial function *(or* ***partial mapping****)*

$^{\forall}a \in A, |f(a)| \leq 1.$

(total) function *(or* ***mapping****)*

$^{\forall}a \in A, |f(a)| = 1.$

Let f is a function and f(a)={b}. Then we write

$f(a) = b.$

restriction *of f to A': f | A' ≡ f': A' → B*

$f : A \to B,\ A' \subseteq A.$ *Then*

$f'(a) = f(a)\ ^{\forall}a \in A'.$

A (total) function f: A → B is

an ***injection*** *(or* ***one-to-one****),*

if $f(a) = f(b) \Rightarrow a = b;$

or $a \neq b \Rightarrow f(a) \neq f(b).$

f is a ***surjection*** *(or* ***onto****) if f(A) = B.*

$(f(A) \subseteq B$ *in general)*

A function that is both injection and surjection

is called a ***bijection*** *(or* ***one-to-one onto****).*

f^{-1} *also is a* ***(bijective)*** *function.*

Fact 1.6

(1) id_A: bijection $A \to A$.

(2) If f: bijection $A \to B$, f^{-1}: bijection $B \to A$.

(3) If f: bijection $A \to B$, g:bijection $B \to C$,
$f \cdot g$(or $g \circ f$): bijection $A \to C$.

*Let $\mathbb{U}$ be a collection of all sets(**universe**). Then*
*a set A in $\mathbb{U}$ is **isomorphic** with a set B in $\mathbb{U}$,*
$A \cong B$, if there is a bijection from A to B.

Fact 1.7 *The set isomorphism $\cong$ is an **equivalent** relation on $\mathbb{U}$.*

cardinal numbers
equivalence classes under set isomorphism
*$[A]_{\cong} \equiv |A|$ the **size** (or **cardinality**) of set A*

*A set is **finite** same size as the set {1, … ,n}*
for some natural number n.
*A set is **infinite** if it not finite.*
*A set is **countable**(or **denumerable**)*
same size of some subset of $\mathbb{N}$
$\mathbb{N}$: set of all natural numbers.
*A set is **uncountable**(or **nondenumerable**)*
if it is not countable.
*A countable infinite set is called **countably infinite**.*

If $^{\exists}$*an* ***injection****(or* ***one-to-one****)* $f: A \to B$,

$|A| \leq |B|$.

If $^{\exists}$*a* ***surjection****(or* ***onto****)* $f: A \to B$,

$|A| \geq |B|$.

If $^{\exists}$*a* ***bijection****(or* ***one-to-one onto****)* $f: A \to B$,

$|A| = |B|$.

Proposition 1.8 *Any countably infinite set is of* ***size*** $|\mathbb{N}|$.

countably infinite set $\{a_0, a_1, \ldots\}$

finite set(countable) $\{a_0, a_1, \ldots, a_n\}$ *for some n.*

Lemma 1.9 *Let* $\{A_n | \ n = 0, 1, \ldots\}$ *be a* ***collection of pairwise disjoint finite*** *sets. Then the set*

$\cup_{n=0}^{\infty} A_n$ *is* ***countable****.*

If A and B are sets, we define

$A^B \equiv \{f | \ f$ *is a function from B to A*$\}$.

$|A^B| = |A|^{|B|}$.

If A is a set, we define

$2^A \equiv P(A) = \{B | \ B \subseteq A\}$ *power set of A.*

$|2^A| = 2^{|A|}$.

$A^{\mathbb{N}}$: ***infinite strings***(*or* ***sequences***) *over A*

$A^{\mathbb{N}}: \mathbb{N} \to A$ ($\{0, 1, \dots\} \to A$)

$|A^{\mathbb{N}}| = |2^{\mathbb{N}}|$.

$f \in A^{\mathbb{N}}$ *and* $f(i) = a_i \in A$, $i = 0, 1, \dots$

$f = (a_0, a_1, \dots)$.

Theorem 1.10 $\{0, 1\}^{\mathbb{N}}$, *the set of all infinite string over* $\{0, 1\}$ *is,* ***uncountable****.*

Proof*.* ***Cantor's Diagonal Argument.***

Assume $\{0, 1\}^{\mathbb{N}}$ *is countable.*

$\{0, 1\}^{\mathbb{N}} = \{f_0, f_1, \dots\}$

where $f_i = (a_{i0}, a_{i1}, \dots)$ *and* $^{\forall} i, j \in \mathbb{N}, a_{ij} \in \{0, 1\}$.

"*diagonal elements*" a_{ii}, $i \in \mathbb{N}$, *we can construct* f

$f = (b_0, b_1, \dots)$, $b_i=0$, *if* $a_{ii}=1$; $b_i=1$, *if* $a_{ii}=0$.

$f(i) = b_i \neq a_{ii} = f_i(i)$ *for any* $i \in \mathbb{N}$. $\therefore {}^{\forall} i \in \mathbb{N}, f \neq f_i$.

$\therefore f \in \{0, 1\}^{\mathbb{N}}$ *but* $f \notin \{f_0, f_1, \dots\}$.

$\therefore \{0, 1\}^{\mathbb{N}} = \{f_0, f_1, \dots\}$ *is a* ***contradiction****.*

$|\{0, 1\}^{\mathbb{N}}| > |\mathbb{N}|$.

$|\{0, 1\}^{\mathbb{N}}|$ *is* ***uncountable****.*

1.2 Languages

*A pair $(M, \cdot)$ is a **algebraic system**, if M is a set and*
$\cdot$ is a binary operation on M.
$\cdot : M \times M \to M$.

*A pair $(M, \cdot)$ is a **semigroup**, if M is a set and*
$\cdot$ is associative binary operation on M.

*An element $e \in M$ is an **identity** of a semigroup $(M, \cdot)$,*
if for all $x \in M$, $e \cdot x = x \cdot e = x$.

***Lemma 1.11** A semigroup has at most one **identity**.*
$e_1 \cdot e_2 = e_2 \cdot e_1 = e_1 = e_2$

*A triple $(M, \cdot, e)$ is a **monoid.***
*(or **semigroup with identity**)*

***Fact 1.12** Let A be a set and let $\cdot$ be a multiplication of relations on A. Then $(2^{A \times A}, \cdot, id_A)$ is a monoid. (Here $2^{A \times A}$ denotes the set of all subsets of $A \times A$, i.e., the set of all relation on A.)*

Let M be a monoid, x an elements of M and n a natural number. The n^{th} power of x, denoted by x^n,
(1) $x^0 = e$;
(2) $x^n = x \cdot x^{n-1}$, for $n>0$.

Extend $\cdot$ to a binary operation on 2^M

$A \cdot B = \{x \cdot y \mid x \in A, y \in B\}$.

Fact 1.13 *Let $(M, \cdot, e)$ be a monoid. Then*

*$(2^M, \cdot, \{e\})$ is an (**induced**) **monoid**.*

If A is subset of M and x is an element of M,
we may write xA in place of {x}A and Ax in place of A{x}.

*A subset A of a monoid M is **closed**, if* $^{\forall}n \in \aleph$,

$^{\forall}x_1, \ldots, x_n \in A \Rightarrow x_1 \cdot \ldots \cdot x_n \in A$.

where $x_1 \cdot \ldots \cdot x_n \equiv e$, if $n=0$.

*A is **positively closed** if the above implication is true for all positive n.*

Fact 1.14 *Let A be a subset of a monoid. Then*

*(1) A is **positively closed** if and only if*

$^{\forall}x, y \in A \Rightarrow x \cdot y \in A$.

*(2) A is **closed** if and only if A is positively closed and contains the identity e.*

Fact 1.15 *Let A be a* ***closed*** *subset of a monoid (M, ·, e). Then (A, ·, e) where · is the restriction of the operation of M to A is also a* ***monoid****.*

(A, ·, e) ***submonoid*** *of (M, ·, e).*

Let A be a subset of a monoid (M, ·, e).

The ***positive closure*** *of A, denoted by* A^+*, is*

$A^+ = \cup_{n=1}^{\infty} A^n$.

The ***closure*** *of A, denoted by* A^**, is*

$A^* = \cup_{n=0}^{\infty} A^n$.

Lemma 1.16 *Let A be a subset of a monoid (M, ·, e). Then* A^+ *is the* ***smallest positively closed subset*** *of M that includes A, and* A^* *is the* ***smallest closed subset*** *of M that includes A. In other words,*

(1) A^+ *is* ***positively closed*** *and* $A \subseteq A^+$.

(2) $A^+ \subseteq B$ *whenever B is a* ***positively closed*** *subset of M such that* $A \subseteq B$.

(3) A^* *is* ***closed*** *and* $A \subseteq A^*$.

(4) $A^* \subseteq B$ *whenever B is a* ***closed*** *subset of M such that* $A \subseteq B$.

Let M be a monoid, and $B \subseteq M$, if $B^* = M$
*B **generates**(or **spans**) M, or*
*B is a **basis**(or **generator**) of M*
If $x_1, \ldots, x_n \in B$, $x_1 \cdot \ldots \cdot x_n \in M$ for some $n \geq 0$.

*B <u>generates</u> M **freely**, if this representation is <u>unique</u>.*
*M is a **free monoid**.*

***Lemma 1.17** Let M be a <u>free</u> monoid. Then*
$\forall x, y, z \in M$.
(1) $zx = zy \Rightarrow x = y$ ***left cancellation***
(2) $xz = yz \Rightarrow x = y$ ***right cancellation***

Let V be a set and n be a natural number.

$V^{\{1, \ldots, n\}}$ *the functions from* $\{1, \ldots, n\}$ *to V.*

$x \in V^{\{1, \ldots, n\}}: \{1, \ldots, n\} \rightarrow V$

(finite) strings *of length n over V*

(sequences) *of elements in V*

If $x \in V^{\{1, \ldots, n\}}$ *and* $x(i) = a_i$, $1 \le i \le n$,

$x \equiv (a_1, \ldots, a_n)$.

If $n=0$, $\{1, \ldots, n\} = \varnothing$. $V^{\varnothing} = \{(a_1, \ldots, a_n) \mid n=0\}$

$(a_1, \ldots, a_n) \equiv \varepsilon$ ***empty string***

$V^{\varnothing} = \{\varepsilon\}$ *sring of length 0 over V*

The set of ***(finite) strings****(or* ***sequences****) over V*

$Strings(V) = \cup_{n=0}^{\infty} V^{\{1, \ldots, n\}}$.

A ***string concatenation*** *operation on Strings(V)*

If $x = (a_1, \ldots, a_m)$, $y = (b_1, \ldots, b_n)$. *Then*

$x \cdot y = (a_1, \ldots a_m, b_1, \ldots, b_n)$.

$$(x \cdot y)(i) = \begin{cases} a_i, & \text{for } 1 \le i \le m \\ b_{i-m}, & \text{for } m+1 \le i \le m+n. \end{cases}$$

Fact 1.18 *For any set V,* $(Strings(V), \cdot, \varepsilon)$ *is a monoid, where* $\cdot$ *denotes string concatenation.*

Lemma 1.19 *Let V be a set and n a natural number. Then*

(1) $(V^{\{1\}})^n = V^{\{1, \ldots, n\}}$.

(2) $(V^{\{1\}})^+ = Strings(V) \setminus \{\varepsilon\}$.

(3) $(V^{\{1\}})^* = Strings(V)$.

(4) The generation of Strings(V) by $V^{\{1\}}$ *is* ***free***.

(Strings(V), ·, ε) is a ***free monoid*** *generated by* $V^{\{1\}}$.

The function g from V to $V^{\{1\}}$ *defined by*

$g(a) = (a), a \in V$, *is a* <u>*bijection*</u>.

(a)	*a string of length 1 over V*
a	$a \in V$

$\therefore$ $V^{\{1\}}$ *and V are* ***identical***.

Theorem 1.20 *Any set can be embedded as a* ***basis*** *into a* ***free monoid***. *Since* $V^{\{1\}} = V$.

(1) $a_1 \ldots a_n$	*for* $(a_1, \ldots, a_n)$,
(2) V^n	*for* $V^{\{1, \ldots, n\}}$,
(3) V^+	*for* $Strings(V) \setminus \{\varepsilon\}$,
(4) V^*	*for* $Strings(V)$.

$(V^*, \cdot, \varepsilon)$ *is a* ***free monoid generated*** *by V.*

Let $x = a_1 \ldots a_n \in V^n$, $n \geq 0$. *Then*

$x^R \equiv a_n \ldots a_1 \in V^n$,

reverse(*or* ***mirror image***) *of x.*

$|x| \equiv n$.

$Prefix(x) = \{y \mid yz = x, \text{ for some } z \in V^*\}$.

$Suffix(x) = \{z \mid yz = x, \text{ for some } y \in V^*\}$.

If k is a natural number. Then

$$k{:}x = \begin{cases} x, & \text{for } |x| \leq k; \\ a_1 \ldots a_k & \text{for } |x| > k. \end{cases}$$

prefix of length of $min\{k, |x|\}$ *of x.*

$x{:}k = (k{:}x^R)^R$.

suffix of length of $min\{k, |x|\}$ *of x.*

The operations ^{R}, *Prefix, Suffix,* $k{:}$, *and* $:k$ *on* $L \subseteq V^*$.

$L^R = \{x^R \mid x \in L\}$

$Prefix(L) = \{x \mid x \in Prefix(y), y \in L\}$

$= \cup_{x \in L} Prefix(x)$

$Suffix(L) = \cup_{x \in L} Suffix(x)$

$k{:}L = \{k{:}x \mid x \in L\}$

$L{:}k = \{x{:}k \mid x \in L\}$

Let M_1 and M_2 be monoids. A function h
from M_1 to M_2 is a ***homomorphism****, if*
(1) $h(x \cdot y) = h(x) \cdot h(y)$, $^{\forall}x, y \in M_1$,
h preserve the ***operation***
(2) $h(e_1) = e_2$.
h preserve the ***identity****.*

A homomorphism that is also bijection is called
isomorphism*.*
Monoid M_1 and M_2 are ***isomorphic****,*
if there is an isomorphism from M_1 to M_2.
$i(M_1) = M_2$ and $i^{-1}(M_2) = M_1$.

Lemma 1.21 *Let h, h': homomorphism $M_1 \rightarrow M_2$,*
B_1 be a basis of M_1. If $h(x) = h'(x)$ $^{\forall}x \in B_1$.
Then $h = h'$.

Lemma 1.22 *Let M_1: free monoid with basis B_1*
M_2: monoid, h: function $B_1 \rightarrow M_2$. Then
$h'(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$, $a_1, \ldots, a_n \in B_1$,
defines a homomorphism h': $M_1 \rightarrow M_2$ such that
$h'(a) = h(a)$ for all $a \in B_1$.

*A **free monoid** $(V^*, \cdot, \varepsilon)$ **generated** by V.*

*A set V is an **alphabet**(or **vocabulary**),*
if it is finite and nonempty.
The elements of an alphabet V are called
***symbols**(or **letters** or **characters**)*
$a \in V$
*A **language** L over V is*
any subset of a free monoid V^*.
$L \subseteq V^*$
The elements of a language L are called
***sentences** of L.*
$x \in L \subseteq V^*$

***Theorem 1.23** If V is any alphabet. Then*
*(1) V^* is a **countably** infinite set.*
*(2) All languages over V are **countable** sets.*

***Fact 1.24** Let V be an alphabet. Then any string w over V can be encoded uniquely as a string w' over {0, 1}. Moreover*
$|w'| = |w|(\lfloor log_2|V| \rfloor + 1)$.

$\|w\| \equiv |w| log_2|V|$ ***norm** of string w over V*

1.3 Random Access Machine

Program *a sequence of instructions*

+

Configuration*(or* ***instantaneous description****)*

$C = (i, M, F_{in}, F_{out})$

i: instruction pointer

M: an infinite sequence of integer (memory)

$(a_1, a_2, \ldots)$

F_{in}*: a finite sequence of integer (input files)*

F_{out}*: a finite sequence of integer (output files)*

Initial configuration for a string $w = a_1 \ldots a_n$

$(1, (0, 0, \ldots), (code(a_1), \ldots, code(a_n)), ())$

Final configuration

i points to ***halt*** *instruction*

Error configuration

the instruction *pointed by i cannot be* ***executed***

invalid address

end of input file

Computation*(or* ***process****)* ***on*** *string w*

a sequence of configurations $(C_0, \ldots, C_n)$

C_{i+1} *is a* ***successor*** *configuration of* C_i

Deterministic
successor configuration is ***unique****, if exists.*
Nondeterministic
successor configuration is ***not unique****.*

Example ***guess*** *a*
Given configuration $C = (i, M, F_{in}, F_{out})$ *where i points the* ***guess*** *a instruction any configuration* $(i+1, M', F_{in}, F_{out})$ *where M' coincides with M in all but* a^{th} *memory location is a legal successor to C.*

A computation is ***correctly terminated****,*
if it ends in a final configuration.
A computation is ***incorrectly terminated****,*
if it ends in an error configuration.

A RAM program ***halts correctly on*** *input w,*
if it ends in a correct termination.
A RAM program ***halts incorrectly on*** *input w,*
if it ends in a incorrect termination.
A RAM program ***loops forever on*** *input w,*
if it there is an arbitrary long computation.

A string w' is an ***output produced for*** *input w,*
if $^{\exists}$*correctly terminated computation on w, and*
the output file represents w'
in the final configuration.

Nondeterminism
"halts correctly", "halts incorrectly", "loop forever"
are ***not mutually exclusive****.*

Let M be ***deterministic*** *and* ***halts correctly*** *for* ***all*** *input*
transformation*:* ***total*** *function* f_M
f_M*: set of input strings* $\rightarrow$ *set of output strings*
f_M*:* ***function defined****(or* ***computed****)* ***by*** *M*
$f_M(w)$
M ***transforms*** *string w into* $f_M(w)$*.*
M ***constructs****(or* ***computes****)* $f_M(w)$ *from w*
Pascal Compiler

*A function f is **recursive**(or **computable**),*
if it is the function defined by some transformation,
i.e., if $f = f_M$ for some transformation M.
determinstic and total function

RAM porgram
Language recognizer
*M **accepts** a string w, if it constructs 1 from w*
***accepting computation** on w*
***language accepted**(or **recognized)** by M*
L(M) = {w| M produce output 1 on w}

M may be nondeterministic,
(1) M accepts w
(2) M does not accept w
(3) M halts incorrectly on w
(4) M loops forever on w

*A language is **recursively enumerable**, if*
it is accepted by some RAM program M.
(note that L is defined nondeterminstically)
*A language is **recursive**, if*
*it is accepted by some **deterministic** program*
*that **halts correctly** for all inputs.*

RAM program
Algorithms with self-explaining high-level language

1.4 Decision Problems

A **decision problem** P **over** alphabet V

partial function from V^* to $\{0, 1\}$

instance of P: $P^{-1}(\{0, 1\})$

yes-instance: $i \in P^{-1}(1) \subseteq V^*$

no-instance: $i \in P^{-1}(0) \subseteq V^*$.

note that $V^* \setminus P^{-1}(\{0, 1\}) \neq \emptyset$.

Language associated with P

$L(P) = \{i \in V^* | \ i \in P^{-1}(1)\}$

Complement of decision problem P over V

$L(\overline{P}) = P^{-1}(\{0, 1\}) \setminus L(P)$

$L(P) \cap L(\overline{P}) = \emptyset$ but $V^* \setminus P^{-1}(\{0, 1\}) \neq \emptyset$

Partial solution to a decision problem P over V

$^{\exists}M$ such that $L(M) = L(P)$

(Total) solution

deterministic and halts correctly.

(totally) solvable(decidable)	total solution
partially solvable(decidable)	partial solution
unsolvable(undecidable)	not solvable

P_{Pascal}: "Is string w a syntactically correct Pascal program?" is solvable.

Lemma 1.25 *Any decision problem that has only finite number of instance is* ***solvable****.*

Proof

If $\{w_1, \ldots, w_n\}$ *is the set of instances of decision problem P in question, let* b_i *denotes the value* ***true****, if* $P(w_i) = 1$*; and the value* ***false****, if* $P(w_i) = 0$*.*

```
boolean procedure M(string w);
    if        w = w1 then return b1
    elseif    w = w2 then return b2
    ...
    elseif    w = wn then return bn
    otherwise                return false
end procedure
```

The solution ***exists****, but it can* ***not*** *be* ***constructed****.*

decision problem on $¿P(w_i) = 1$

The number of distinct functions P from V^* *to* $\{0, 1\}$ *is uncountable.*

The number of all program is countable.

$\therefore$ ***not all*** *the decision problem is partially solvable.*

Lemma 1.26 *For any alphabet V, there are decision problems over V that are* ***not partially solvable****.*

halting problem

P_{halt}: "Does procedure M halt correctly on w?"

boolean procedure *M_{halt}(**procedure** M, **string** w);*

begin

M(w);

*return **true***

***end**.*

Partial solution on P_{halt}.

*M_{halt} return **true**, iff M correctly halts on w*

M_{halt} halts incorrectly, iff M halts incorrectly

M_{halt} loops forever, iff M loops forever

procedure M_{halt} is determnistic,

but the above solution is partial.

*P_{halt} is **not totally** solvable.*

Proof

P'_{halt}: "Does procedure M halt correctly on M'?"

*P' is called a **restiction** of P, if*

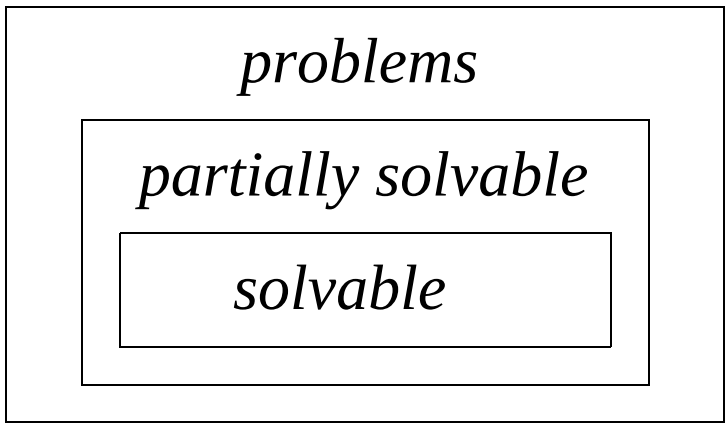
the instance of P' is a subset of the instance of P

*P'_{halt} is a **restriction** of P_{halt}, $(M\#M' \subseteq M\#w)$*

Assume that P'_{halt} is (totally) solvable.
P'_{halt} has a solution $M'_{halt}(M, M')$, which
*returns **true**, M halts correctly on M',*
*returns **false**, otherwise.*

***procedure** M̂(**procedure** M);*
if** $M'_{halt}(M, M)$ **then loop forever fi.
M̂ halts correctly on M, iff M does not halt correctly on input M, but this is not true for M = M̂.
∴ M'_{halt} does not exist, and P'_{halt} is not solvable.

***Theorem 1.27** There exist partially solvable problems that are not totally solvable.*



Lemma 1.26
existance of non partially solvable
Theorem 1.27
exsistance of not solvable but partially solvable

*P_{exhalt}: "Does procedure M halt correctly on **some** w?"*

P_{exhalt} is partially solvable but not solvable.

```
boolean procedure M_exhalt(procedure M);
    begin string w;
        guess w;
        M(w);
        return true
    end.
```

M_{exhalt}: nondeterministic partial solution
determinstic version much harder

Proposition 1.28 *Any partially solvable decision problem has a deterministic partial solution.*

determinism does not reduce the partially solvable classes.

Let P_1 and P_2 be decision problems. A ***deterministic*** *RAM program M that halts correctly on all input is a* ***reduction*** *of P_1 to P_2 if M transforms*

all yes-instance of P_1 to yes-instance of P_2, and

all no-instance of P_1 to no-instance of P_2.

If there exists a ***reduction*** *of P_1 to P_2.*

P_1 ***reduces****(is* ***reducible****)* ***to*** *P_2, written $P_1 \leq P_2$.*

Fact 1.29 *Given a decision problem P, any* ***restriction*** *P' of P* ***reduces*** *P.*

Proof

A program that copies the input to output.

P'_{halt} reduces to P_{halt}.

*P_{accept}: "Does boolean procedure M return **true** on w?"*

P_{halt} reduces to P_{accept}.

program T: transform (M, w) into $f_T(M, w)$

*$f_T(M, w)$ = (**boolean procedure** M'(**string** w);*

*begin M(w); **return true end**, w)*

*M' returns **true**, iff M halts correctly on input w.*

(M, w) is a yes-instance of P_{halt}, iff

$f_T(M, w)$ is a yes-instance of P_{accept}.

$\therefore P_{halt}$ reduces to P_{accept}.

***Lemma 1.30** Let P_1 and P_2 be decision problems such that P_1 reduces to P_2. Then*

*(1) If P_2 is **solvable**, then so is P_1.*

*(2) If P_2 is **partially solvable**, then so is P_1.*

*(3) If P_1 is **unsolvable**, then so is P_2.*

*(4) If P_1 is **not partially solvable**, then neither is P_2.*

***Proof** (3) and (4) are restatement of (1) and (2).*

If M is a reduction of P_1 to P_2 and M_2 is a (partial) solution to P_2.

***boolean procedure** M_1(**string** w)*

begin return** $M_2(M(w))$ **end.

M_1 is a (partial) solution to P_1.

1.5 Computational Complexity

measure of the computational resources

computation time

space used by the program

computational complexity

time complexity

space complexity

uniform cost criterion

logarithmic time cost criterion

$(\lfloor \log a \rfloor + 1) + (\lfloor \log \max\{1, |v|\} \rfloor + 1)$

value v is stored in or retrieved form a memory location a

logarithmic space cost criterion

$\lfloor \log \max\{1, |n|\} \rfloor + 1$ … *storing n*

Let $\mathbb{C} = (C_0, \ldots, C_n)$ *be a computation of a RAM.*

uniform cost time complexity *of* $\mathbb{C}$

n

logarithmic cost time complexity *of* $\mathbb{C}$

$\sum t_i$ t_i *is a logarithmic time complexity for* C_i

uniform cost workspace complexity *of* $\mathbb{C}$

$\max\{|C_0|, \ldots, |C_n|)$

logarithmic cost workspace complexity *of* $\mathbb{C}$

$\max\{\|C_0\|, \ldots, \|C_n\|\}$

space complexity *of* $\mathbb{C}$

workspace complexity of $\mathbb{C}$ *+ input output space*

Let M be a transformation and w be a string and $\mathbb{C}$ the terminated computation of M on w.
M ***transforms*** *string w* ***into*** *string $f_M(w)$ (or M* ***constructs*** *or* ***computes*** $f_M(w)$ ***from*** *w)*
(1) ***in time*** *t if the time complexity of $\mathbb{C}$ is at most t,*
(2) ***in space*** *s if the space complexity of $\mathbb{C}$ is at most s,*
(3) ***in workspace*** *s if the workspace complexity of $\mathbb{C}$ is at most s.*

Let T and S be a function from natural number to the set of non-negative real numbers.
M in T(n) ***time-bound****(or* ***runs in time*** *T(n)),*
if for all natural number n and string w of ***size*** *n, M transforms w into $f_M(w)$ in time T(n).*
*M is S(n) (****work****)****space-bound****, if ….*

uniform cost	*length of w*
logarithmic cost‘	*norm of w*

time complexity *of M*	*least function among T’s*
space complexity *of M*	*least function among S’s*
workspace complexity *of M*	*least function among S’s*

language recognizer *M*
M accept string w ***in time*** *t (or* ***(work)space*** *s)),*
if M has an accepting computation of time complexity at most t(...).

M is in T(n) ***time bounded****,*
in S(n) ***(work)space bounded****,*

time complexity*,*
(work)space complexity*,*

A decision problem P
P is ***solvable in nondeterministic time*** *T(n),*
if it has a T(n) time bounded ***partial solution****.*
P is ***solvable in nondeterministic (work)space*** *S(n),*
if it has a S(n) (work)space bounded ***partial solution****.*

P is ***solvable in deterministic time*** *T(n),*
if it has a ***deterministic*** *T(n) time bounded* ***partial solution****.*
P is ***solvable in deterministic (work)space*** *T(n),*
if it has a ***deterministic*** *S(n) (work)space bounded* ***partial solution****.*

Proposition 1.31 *Let f be a function on the set of natural numbers. If a decision problem is solvable in nondeterministic time f(n) or in nondeterministic (work)space f(n), then it is* ***solvable****.*

Let f and g be function from the set of natural numbers to the set of real numbers.
g(n) is ***order*** *f(n), written g(n) = O(f(n)),*
if $^{\exists}c, n_0 \in \mathbb{N}$, $g(n) \le cf(n)$, $^{\forall}n \ge n_0$.
g(n) ***does not glow much faster*** *than f(n).*

Lemma 1.32 *Let* $f, g: \mathbb{N} \to \mathbb{R}$, $a, b \in \mathbb{R}$.
(1) af(n) + b is O(f(n)).
(2) f(n) + g(n) is O(max{f(n), g(n)}).

P_{pal}*: "Is string w a palindrome?"*
deterministic uniform cost time O(n),
deterministic logarithmic cost time O(n log n)
cost (work)space O(n)
$w = a_1 \ldots a_n$, $^{?}a_i = a_{k-i+1}$, $i = 1, \ldots, \lfloor k/2 \rfloor$.
O(k log|V|) logarithmic space cost
O(k) uniform cost
k reads and $\lfloor k/2 \rfloor$ *comparisons*
O(k(log k + log |V|)) logarithmic time cost
size of O(log(|V|)) and addresses of O(log k)
for each O(k) reads and comparisons

Lemma 1.33 *Let M be a RAM program with time complexity T(n), space complexity S(n), and workspace complexity S'(n).*

(1) S'(n) is O(T(n)).

(2) S(n) is O(max{n, T(n)}).

Proposition 1.34 *Let P be a decision problem.*
If P is solvable in nondeterministic time T(n),

$^{\exists}c$*, a constant depending only on P,*

P is solvable in ***deterministic*** *time* $O(c^{T(n)})$.

Proposition 1.35
Let M be a deterministic language recognizer.
If M runs in workspace $S(n) \geq \log n$. *Then*

$^{\exists}c$*, a constant depending only on M,*

M runs in time $O(c^{S(n)})$.

A function f is ***space-constructible****,*

if it is a logarithmic cost workspace complexity of some RAM program.

Fact 1.36
The identity function $id_{\mathbb{N}}$ *is* ***space-constructible****. Furthermore any constant function on* $\mathbb{N}$ *is* ***space-constructible****.*

Proposition 1.37 *If f and g are space-constructible.*
(1) nf+m and ⌊log f⌋ are space-constructible.
(2) fg and f^g are space-constructible.

Proposition 1.38 *(Savitch's Theorem)*
Let S be a space-constructible, $^\forall n$.∋. S(n) ≥ log n.
If a decision problem is solvable in ***nondeterministic*** *workspace S(n), it is solvable in* ***deterministic*** *work-space $O(S(n)^2)$.*

Given a partial solution to some decision problem P.
derive a partial solution for ¬P, the complement of P.
it is ***not*** *possible.*
yes-instance ⇏ rejection of no-instance
But, partial solution for P
deterministic*, T(n) time-bound,*
T is easily computable
the solution to P ***carries over*** *¬P.*

Theorem 1.39 *Let M be a deterministic recognizer*
runs in time T(n), space S(n), workspace S'(n),
T(n) is computable in time O(T(n)), space O(S(n))
L(M) has a ***deterministic*** *recognizer M'*
halts correctly *in … O(T(n)), O(S(n)), O(S'(n)).*
Proof *Augment M with counting number of instructions.*
M' write 0, when the counter exceeds T(n).

Corollary 1.40 *If a decision problem P is solvable deterministic time T(n), deterministic space S(n), deterministic O(T(n)) time-bounded, O(S(n)) ...*
total solution,
if T(n) is computable in O(T(n)) and O(S(n)).

Theorem 1.41 *If a decision problem P is solvable in deterministic time T(n),*
¬P is solvable in deterministic time O(T(n))
provided that T is computable in time O(T(n)).
If P is solvable in deterministic (work)space S(n),
¬P is solvable in deterministic (work)space O(S(n))
provided that S is computable in space O(log S(n)).
Proof *Corollary 1.40 and Proposition 1.35.*

Let M be a reduction of a decision problem P_1 to P_2.
If M runs in time O(T(n)) runs in time f(n),
M is an f(n) ***time reduction of*** *P_1 to P_2.*
P_1 ***reduces in time*** *f(n)* ***to*** *P_2.*
P_1 is f(n) ***time reducible to*** *P_2.*
An f(n) time reduction is a ***polynomial time reduction***,
if f(n) is $O(n^k)$ for some constant k.
P_{halt} is a ***polynomial time reduction*** *of P_{accept}.*
Two decision problems are ***equally hard***,
if they reduce to each other in polynomial time.

Lemma 1.42 *Let P_1* ***reduces in time*** *$f(n)$* ***to*** *P_2. Then*
(1) If P_2 is solvable in (non)deterministic time $T(n)$,
P_1 is solvable in (non)deterministic time $O(T(n))$.
(2) If P_2 is solvable in (non)deterministic (work)space $S(n)$,
P_1 is solvable in (non)deterministic (work)space $O(S(n))$.

uniform cost criterion
logarithmic cost criterion

reductions between decision problem
deriving lower bounds for problem complexity
uniform cost polynomial
logarithmic cost polynomial

uniform cost complexity $O(f(n))$
logarithmic cost complexity $O(f(n(\log n)^k))$
for some constant $k \geq 0$.

1.6 Rewriting Systems

*A **rewriting system**(or **semi-Thue system**) is a pair (V, P), where*

V: an alphabet

P: a finite relation on the free monoid V^*.

*$(\omega_1, \omega_2) \in P$, $\omega_1 \rightarrow \omega_2$, **rule**(or **production**)*

left(right)-hand side of the rule

***rewrite** strings in V^* **derive** rule*

$\gamma = \alpha\omega_1\beta \Rightarrow \alpha\omega_2\beta$, $\omega_1 \rightarrow \omega_2 \in P$, $^{\forall}\alpha, \beta \in V^*$

γ ***(directly) derives*** $\alpha\omega_2\beta$

*P induces an **infinite** relation on V^*.*

$\gamma_1, \gamma_2 \in V^*$ are related if and only if

γ_1 can be rewritten as γ_2

using some sequence or string of rules

Let G = (V, P) be a rewriting system.

***derives** using rule $r \in P$, $\Rightarrow_G^r$ (or $\Rightarrow^r$ for short)*

$\Rightarrow_G^r = \{(\alpha\omega_1\beta, \alpha\omega_2\beta) \mid r = \omega_1 \rightarrow \omega_2 \in P, \alpha, \beta \in V^*\}$

$\gamma_1 \Rightarrow_G^r \gamma_2$ γ_1 ***(directly) derives*** γ_2

using rule r in G

*rule r is **applicable** to γ_1*

*can be **applied** to γ_1*

***derives** using rule string $\pi \in P^*$, $\Rightarrow_G^{\pi}$ (or $\Rightarrow^{\pi}$ for short)*

Let $\pi \in P^*$. Then we define

$\Rightarrow_G^{\varepsilon} = id_{V^*}$;

$\Rightarrow_G^{\pi} = \Rightarrow_G^{r} \Rightarrow_G^{\pi'}$ *where* $r \in P, \pi' \in P^*, \pi = r\pi' \in P^+$.

If $\gamma_1 \Rightarrow_G^{\pi} \gamma_2$, γ_1 ***derives*** γ_2 *using rule string* π *in* G

rule string π *is* ***applicable*** *to* γ_1

$\Rightarrow_G = \cup_{r \in P} \Rightarrow_G^{r}$

If $\gamma_1 \Rightarrow_G \gamma_2$, γ_1 ***directly derives*** γ_2 *in* G

If $\gamma_1 \Rightarrow_G^{*} \gamma_2$,, γ_1 ***derives*** γ_2 *in* G

γ_2 *is a* ***sentential form*** *of* γ_1.

reflexive trasitive cloure *of* $\Rightarrow_G$.

A string sequence $(\gamma_0, \ldots, \gamma_n)$, $n \geq 0$,

derivation *of* ***length*** n *of* γ_n *from* γ_0 *in* G, *if*

a path of length n *from node* γ_0 *to* γ_n *in*

the directed graph $(V^*, \Rightarrow_G)$; *or*

If $n \geq 0$, $\gamma_i \Rightarrow_G \gamma_{i+1}$ $0 \leq i < n$.

Fact 1.43 *A string* γ_1 *derives* γ_2 *in* G *if and only if* γ_2 *has a derivation from* γ_1 *in* G.

Example

$G_{match} = (\{S, 0, 1\}, \{S \to \varepsilon, S \to 0S1\})$,

where $r_1 = S \to \varepsilon$, $r_2 = S \to 0S1$.

$\alpha S\beta \Rightarrow^{r_1} \alpha\beta$ *for* $\alpha, \beta \in \{S, 0, 1\}^*$

$\alpha S\beta \Rightarrow^{r_2} \alpha 0S1\beta$ *for* $\alpha, \beta \in \{S, 0, 1\}^*$

$S \Rightarrow^{r_2} 0S1 \Rightarrow^{r_2} 00S11 \Rightarrow^{r_1} 0011.$

$S \Rightarrow^{r_2} 0S1 \Rightarrow^{r_2} 00S11 \Rightarrow^{r_2} \ldots \Rightarrow^{r_2} 0^nS1^n \Rightarrow^{r_1} 0^n1^n.$

$S \Rightarrow^{n\cdot(r_2)r_1} 0^n1^n$, *for* $n \geq 0$.

$\therefore S \overset{*}{\Rightarrow} 0^n1^n \; n \geq 0$

proof ***induction*** *on the length of the rule string*

Fact 1.44 Principle of Proof by Induction

Let P(n) be a statement that is either true or false depending on the natural number n. Assume the following statements are true.

(B) $P(0)$.

(I) $^{\forall} n>0, P(0), \ldots, P(n-1) \Rightarrow P(n)$.

or ***(I')*** $^{\forall} n>0, P(n-1) \Rightarrow P(n)$.

Then P(n) is true for all natural number n.

(B) base case

Proof P(0).

(I) induction step ***induction hypothesis***

Proof $^{\forall} n>0, P(n-1) \Rightarrow P(n)$.

*The **size** of rewriting system G(V, P), |G|,*

$|G| = max\ (\Sigma_{\omega 1 \to \omega 2 \in P}\ |\omega_1\omega_2|, |V|)$.

*The **norm** of rewriting system G(V, P), ‖G‖*

$\|G\| = |G|\ log_2|V|$

Lemma 1.45 *Any rewriting system G = (V, P) can be encoded as a binary string of length O(‖G‖).*

Proof.

Let # ∉ V. Then

V can be represented as #α# where α contains exactly one occurrence of each symbols in V.

Any rule $\omega_1 \to \omega_2$ *in P can be represented as the string* $\#\omega_1\#\omega_2$.

Rewriting System *Computer*

string rewriting *instruction*

unordered set of instructions from which any (applicable) instruction can be selected at any moment

sentential form *configuration of computer*

initial sentential form *input string*

intermediate s.f. *current config.*

final sentential form *output string*

Let $D = (\gamma_0 , \dots , \gamma_n)$ be a derivation of length n.

***time complexity** of D* $TIME(D)$

n

***space complexity** of D* $SPACE(D)$

$max\{|\gamma_0|, \dots , |\gamma_n|\}$

Let γ_1 and γ_2 be string such that γ_1 derives γ_2 in G.

$TIME_G(\gamma_1, \gamma_2)$

$\equiv min\{TIME(D)|$ *D is a derivation of* γ_2 *from* $\gamma_1\}$

Fact 1.46

$TIME_G(\gamma_1, \gamma_2) = min\{n \geq 0 | \gamma_1 \Rightarrow^n \gamma_2\}$

$SPACE_G(\gamma_1, \gamma_2) = min\{SPACE(D)|$ *D is a derivation of* γ_2 *from* $\gamma_1\}$

γ_1 ***derives*** γ_2 *in* ***time*** *t, if* $TIME(\gamma_1, \gamma_2) \leq t$.

γ_1 ***derives*** γ_2 *in* ***space*** *s, if* $SPACE(\gamma_1, \gamma_2) \leq s$.

γ_1 ***derives*** γ_2 ***simultaneously*** *in* ***time*** *t and in* ***space*** *s.*

For G_{match},

$TIME(S, 0^n1^n) = n+1 \quad {}^\forall n \geq 0$

$SPACE(S, 0^n1^n) = 2n + 1 \quad {}^\forall n \geq 0.$