Regular language --- finite automata

Context-free language --- pushdown automata

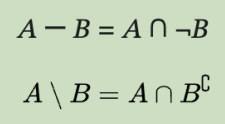
No context free language --- turing machine

Set / subset notation / proper subset（真子集）

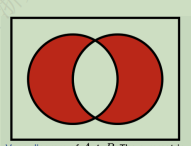
Union并

Intersection 交

Difference 差（两种表示方法）



Symmetric difference（对称差集，用 △ 表示）



Complement 补

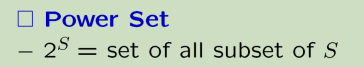
Commutative law 交换律

Idempotent law 幂等律

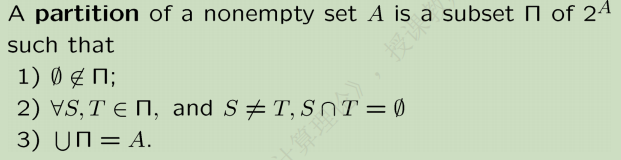
572bff60ace0ac655c6ef78ab1643e2Absorption law 消解律

6fb2be8deed59cb0542c5e8bb22655b

Power set



Partition（空集不能单独成列）



{ {1,2}，{3}，{4,5} } 常被写为 1 2 | 3 | 4 5

Ordered pair（有序对）

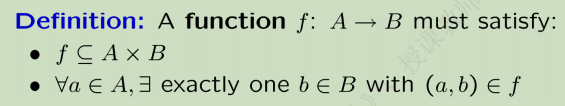
2f2d833558feb1b2aae0aeaf5e57b52Binary relation ---- inverse

Composition

domain set of the input values（需要在有序对中出现）

Range set of the output values（需要在有序对中出现）

Function



Injective function（one-to-one function）domain range一一映射

Surjective function（onto function） range全部被映射

Bijective function（one-to-one + onto）一一映射且满射

40e877ead499720c61978d05f58755d

Reflexive 自反性

Symmetric 对称性 （a，b）属于，（b，a）也属于

Anti-symmetric 反对称性（a，b）属于，（b，a）不属于

可以既不是symmetric也不是Anti-symmetric（都要求任意）

Transitive

满足自反性、对称性、传递性 ----> equivalence class

满足等价关系的无向图中，Cluster表示图中最小的等价类

满足等价关系的等价类集合是该无向图的一个partition

Partial order（偏序），满足自反性、传递性、反对称性

Least：对于任意其他元素，都比它“大”（唯一）

Minimal：没有人比它“小”，可能无法比较（可存多个）

Greatest：对于任意其他元素，都比它“小”

Maximal：没有人比它“大”，可能无法比较（可存多个）

Total order / linearly order（strongly connected）

在偏序的基础上，任意一对elements之间都可以比较

两个集合时是equinumerous（等势？），如果两个集合间存在一个bijection

集合的cardinality表示为集合中的元素个数

Finite set（countable set）自然数集是可数的

可数无穷个可数无穷集合的并还是可数无穷

N\*N是可数无穷（两个bijection） ---> 有理数集是可数无穷

Infinite set（uncountable set）

实数集

Continuum hypothesis：整数集与实数集之间不存在其他基的个数

Cantor’s theorem：任何power set of A一定有比A更大的基

自然数集的power set是不可数无穷集合

Mathematical induction

Pigeonhole principle（可以用数学归纳法证明）

Diagonalization principle （证明整数集合与实数集合个数不一样）

Closure（闭包）满足指定关系且包含R的最小集合

transitive的闭包常表示为R+

Reflexive与transitive的闭包常表示为R\*

Alphabet：finite set of symbols（一个集合）--- 空集是一个字母表

String finite symbol sequence

Concatenation：string的连接符号

空串也属于string

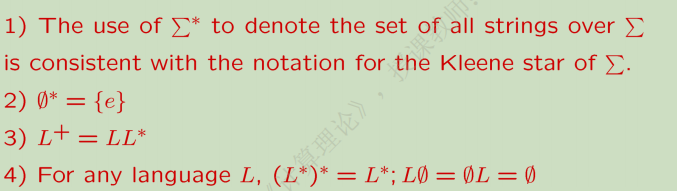
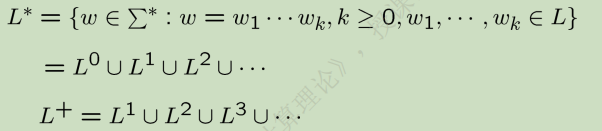
若alphabet有穷，则alphabet\* 为可数无穷，alphabet\*表示alphabet上的所有string，包括空串

String与空串相连为自身

Language：set of strings（属于alphabet\* 的子集，集合的集合）

空集是language，空集不能是alphabet\*

Language可以是有穷的也可以是无穷的（L0就是空串）

ed3e07b3a9c0d147d8ee95a05da94f9

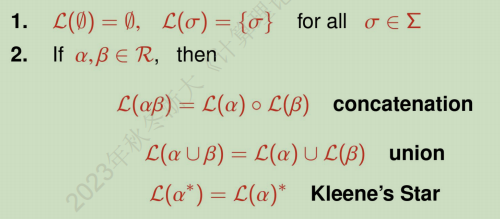
可以把L+看做L在连接运算下的闭包，L+可以含有e（空字符串）

Finite representation of language

Regular expression

空集和单元素集合是regular expression

并集，连接，kleene’s star不影响regular expression

Regular expression ---> regular language

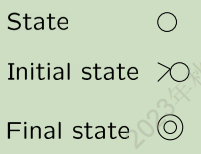
可以借此把regular expression转变为regular language

Regular expression：连接的交换律不成立

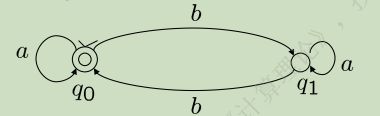
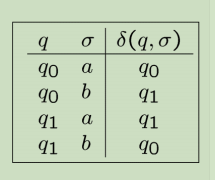
4568020a9019ea91632269dbc4fdc41可用于适当化简：

Deterministic finite automata的图示

状态表示



两种刻画方式（图 + 关系描述）



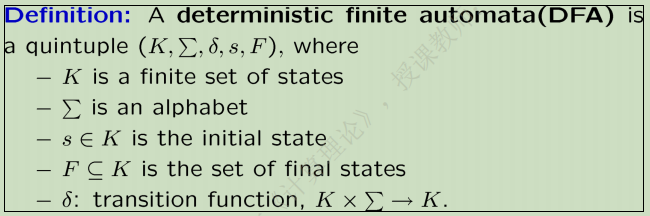
Deterministic finite automata（DFA）的定义，五元素：

K：有限的状态数

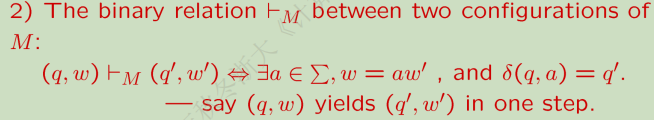
∑：字母表

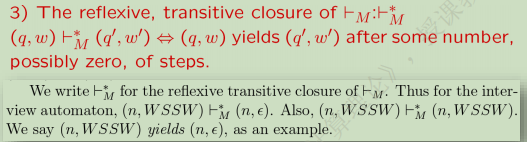
S：初始状态

F：final state

δ ：transition function（注意是function 一一对应）

若将终止状态与一般状态取反，则表示完全相反的regular language

Configuration ---> any elements of K \* ∑ （描述当前的一个状态）

7286d3d891e742de4a365541c220c41355df6076d6a2506f7210e5094d609b

Nondeterministic finite automata（NFA）：五元素

K：有限状态

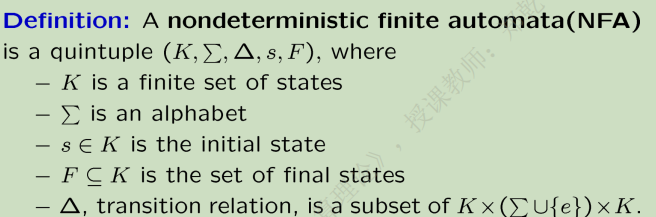
∑：可以接受的字母表

S：初始状态

F：final状态

△：transition relation（注意是relation，可能存在多个映射关系）

（其余configuration / binary relation与DFA定义相似）



DFA与NFA 等价

证明的两个difference

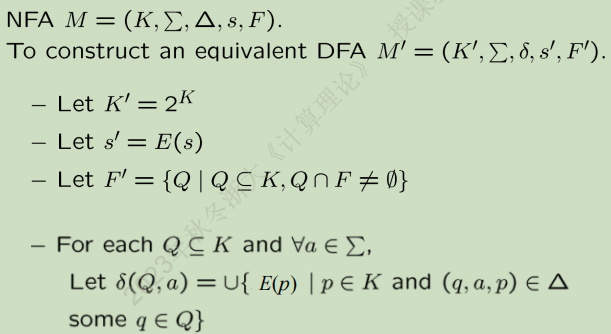
1. 有e的参与 --- 组合state，合并为新的state

含有则为Initial state / final state

1. Relation与function的区别 --- 相同输入引导的state合并为新的state

基于NFA构造DFA

先构造initial state，再对每个已有的state进行扩展，再考虑state的变化情况（新状态包含空集）

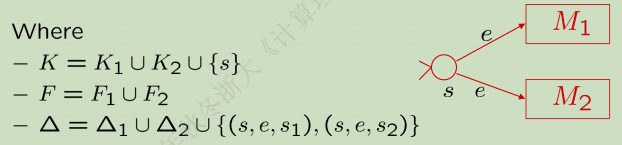


（证明crazy）

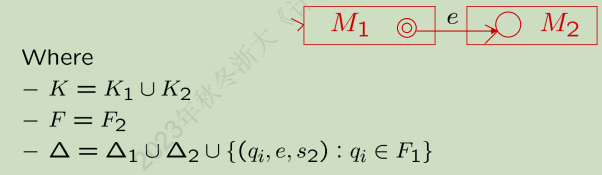
Language is regular iff it is accepted by a finite automaton

Regular language（闭包）

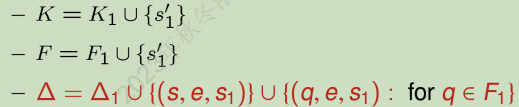
Union

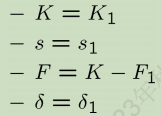


Concatenation

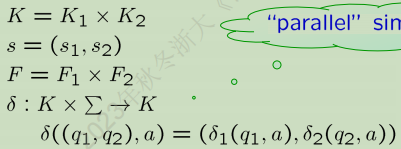


Kleene’s star（需要添加新的initial state）



Complementation

Intersection（类似于并行计算，均被接受）

84a945866d2d1456286e10f99115956

Regular language与FA一一对应

从 finite automata转变为regular expression

需要先添加一个初始状态与终止状态（初始状态不入边，终止状态不出边）

定义generalized automaton，不仅单步可以接收字符，单步也可接收字符串

不停的化简，减少图中的state数量

Regular

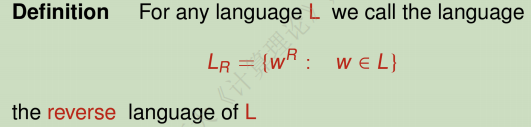
Accepted by FA

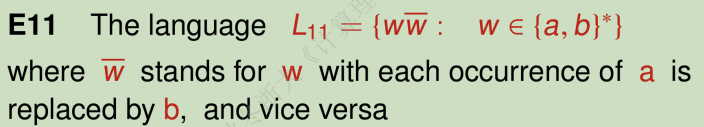
Specified by regular expressions

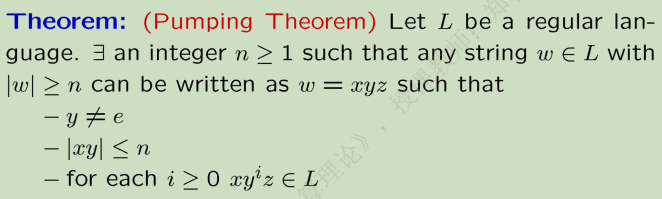
Closure properties of regular language

判断language是不是regular的，先单独判断，再使用闭包技巧判断

∑alphabet上的regular language是无穷可数的

↓ 反向构造，把终止状态改为起始状态，边全部反向即可



n表示FA的状态数（-1 ?），一定有状态重复

Reprise 重复部分

每个DFA定义了unique的language，但是一个language可以对映多个DFA

Context-free language（上下文无关语言）

感觉automata与graph representation还是有一定的区别的

DFA只有recognition的功能，不可以generation

Derivation 起源，派生（sequence of rule application）

Substitution 取代

Implicit 隐含的

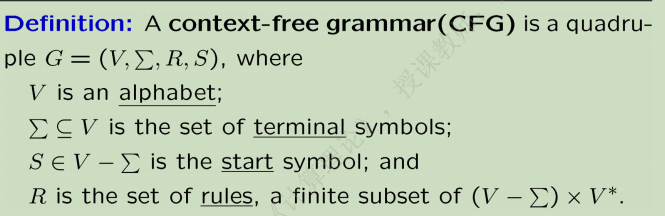
上下文无关语言 --- 四元组

V： alphabet（涵盖所有symbols，一般不为空集）

∑： terminal symbols（属于V，可以为空集）

R：生成rules（substitution rules --- productions）

S：start symbols（唯一）



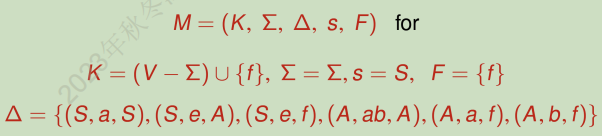
DFA可以接受的语言均可以由CFG生成

用state代替variables，可以把grammar转变为diagram（对映FA）

单步state转换可以接收多个字符

b7da9f08f26d2601ef60ed1ecbc5820

根据context-free grammar construct a FA

增加一个final state，其余继承context-free grammar，在△中将目的（状态）e转变为final state即可

All regular language are CFL（通过FA建立CFG）

空集，单元素集合是 context-free的

CLF在union、concatenation、kleene star闭包

CFL被push down automata接受

Generated by grammar，accepted by automata

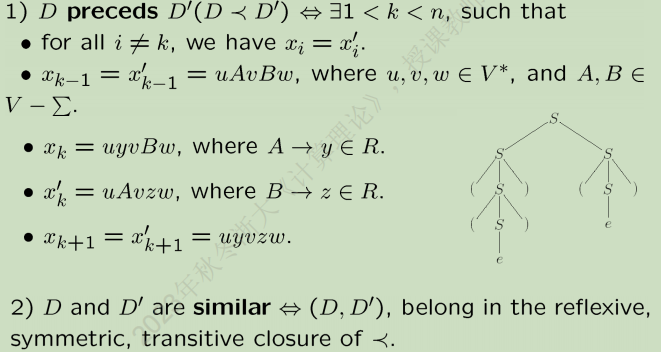
Parse tree（yield of the parse tree）

Terminal symbols can occur only at the leaves

Variables can occur only at the internal nodes

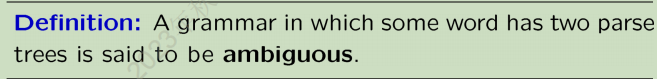
相同的generate结果，可能对映不同的parse tree

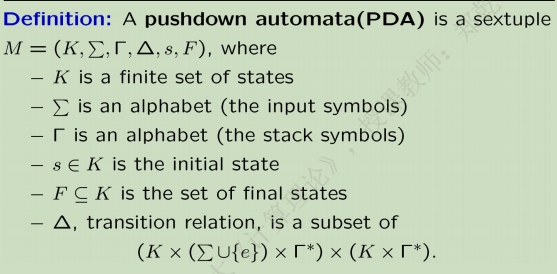
称这些有相同parse tree的derivation为similar

左侧variable先转变为terminal symbol ---> preced（在前）

只有相差一步的derivation才可以使用上述符号

Ambiguity



Pushdown automat（PDA）

六元素（相较于FA需要协调stack中的元素）

K：states

∑：input symbols

倒L：stack symbols

S：initial state

F：final state

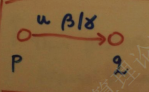
△：transition relation （（q,u,a），（p,e））接收u，从state q转变为state p，stack顶中元素从a转变为e

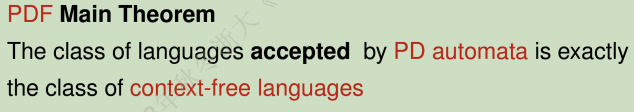
PDA接受的条件：stack为空，state为final state，tape为空

Simultaneous 同时的

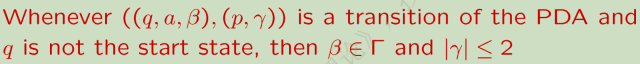
b72586c9803917da690951b4b51c08b

State diagrams（state、non-terminal symbol、stack顶部元素的变化）

CFL generated by CFG（grammar），accepted by PDA（automata）



Simple PDA

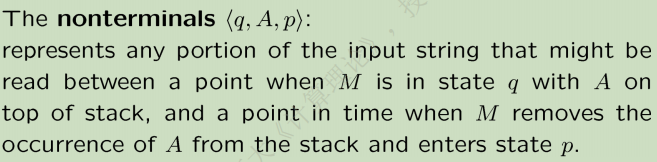


Simple PDA每次只处理stack顶部的元素（注意β一般不为e）

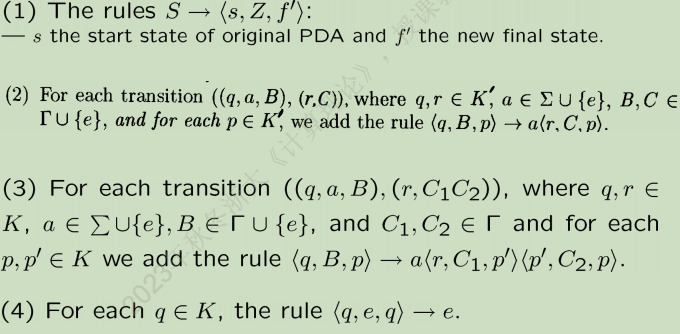
增加一个新的final state和initial state，以及stack中的一个底部元素

拆解stack顶部的多元素计算 ---> 加入新状态，每次仅处理一个元素（每一步stack处理结果可以为e）

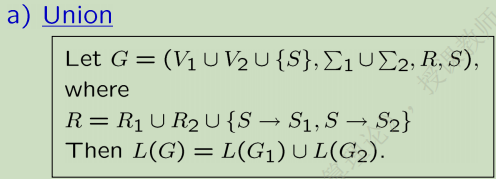
Stack中代替元素超过俩 ---> 每次仅添加一个元素（每一步stack处理源数据可以为e）

处理β为e的情况，向stack中添加共同的元素（可约元素）

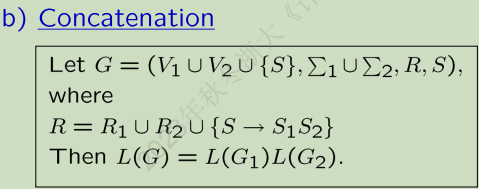
由simple PDA生成CFL的四条法则



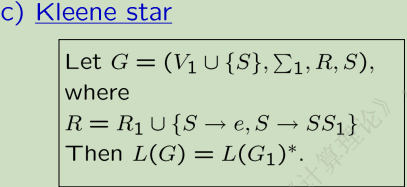
CFL的闭包（没有补和交集）

Union

Concatenation

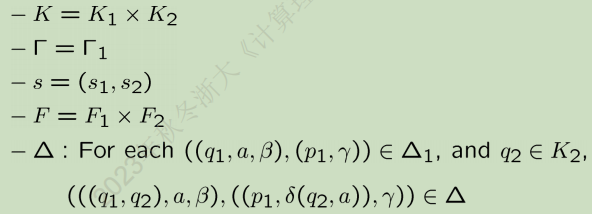


Kleene star

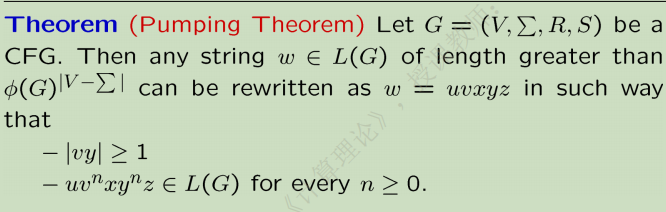


交集，补集不闭包例子（闭包要求是deterministic）：

ab3f7386c0eb7857d651322e587e8d6The intersection of a CFL with a regular language is a CFL（△的改变为关键变化）



CFL的泵定理，其中 |vxy| <= p（给定字符串长度都大于等于p）

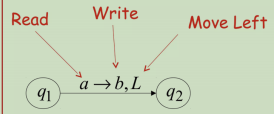


Turing machine

Write / read / move

Unbounded length

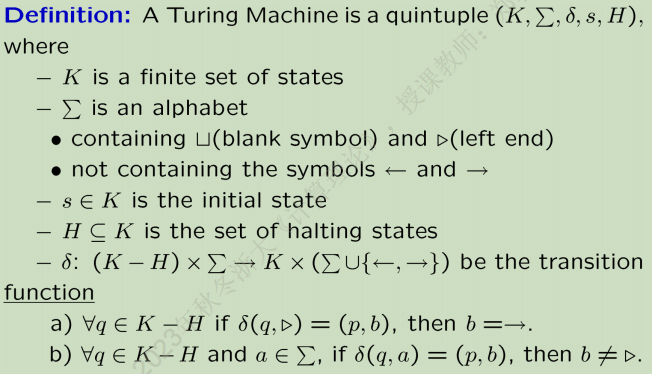
Blank symbol

Turing machine diagram（示意）

Turing machine are deterministic

No possible transition ---> halt / reject

Turing machine 五元组



K：finite state

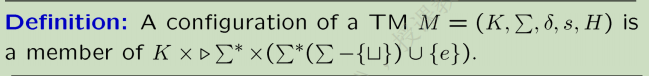
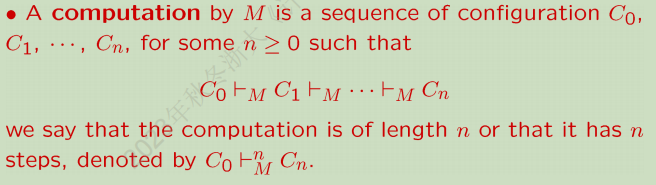
∑：alphabet（包括blank symbol和left end符号，不包括左移右移）

S：initial state

H：halting state

δ ：transition function

↓ K表示当前的状态及header位置信息



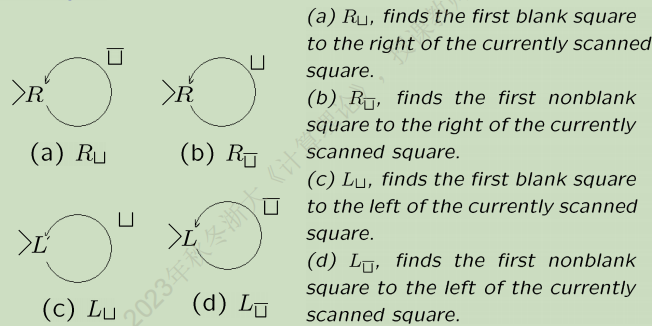
Accepting configuration

Rejecting configuration

L简化左移操作

R简化右移操作

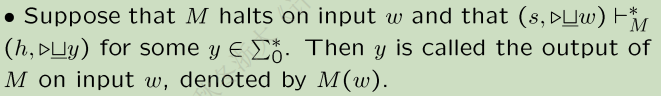
a简化将当前header指向的square改变为a的操作

一些其他的化简表示

图灵机可以接受的语言为recursive language（可递归语言）

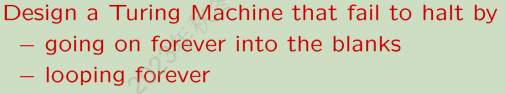
图灵机和递归可枚举语言一一对应

Turing machine用于compute functions（computable）

可被Turing machine计算的function is recursive

Turing machine semidecides（半决定）L，只有接受L时才halting，则称该language为recursively enumerable language（递归可枚举语言）

一定有TM对应的semidecide language但不一定有对应的decide language

93467098219da0b66c07a8ae5955f83

Recursive language闭包

Complement

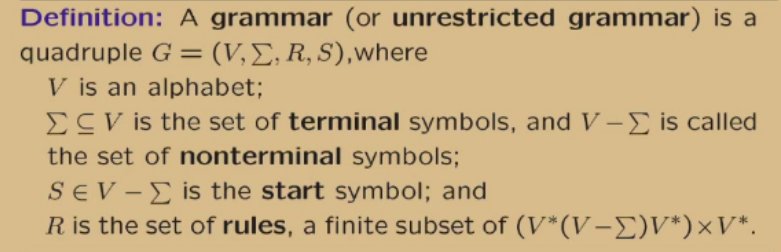
Union

Intersection

若language是recursive language，则它也为recursively enumerable（recursive是recursively enumerable的真子集）

若language的complement和本身都是recursively enumerable的，那它则为recursive language

Grammar（unrestricted grammar）



V：alphabet

∑：terminal symbols

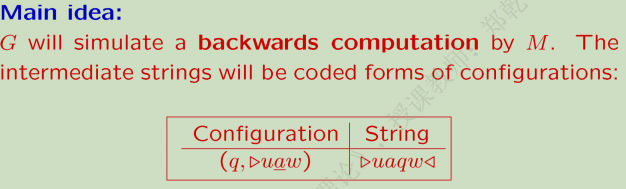
S：start symbols

R：rules（区别于context-free-grammar）

CFG也是一种grammar

Grammar 与recursively enumerable等价

Grammar 与 turing machine等价

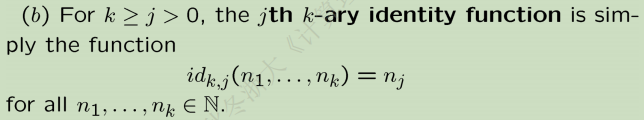


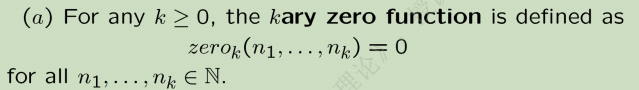
Grammatically computable iff recursive

Recursive由turing machine是否可以计算决定

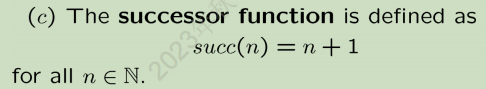
Grammatically computable由grammar是否可以computes决定

Primitive functions（basic function / initial functional）

Projection function（输入k个参数，返回第j个）

Zero function（一直返回0）

Successor function（输入n，返回n + 1）



Two operation：composition and recursive

Primitive recursive function

All basic function，可由two operation组合得到

All constant functions and the sign function are primitive recursive function

Plus、mult、exp函数

m ~ n = max{ m - n, 0 }

Predecessor function：输入为0时，输出为0；输入为 n + 1时，输出为n；

Primitive recursive predicate（primitive recursive function only takes values 0 and 1）

Iszero（n）：仅输入为0时输出1

Positive（n）= sgn（n）

☆Greater-than-or-equal（m,n）= iszero（n ~ m）

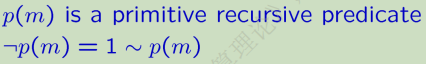
Less-than（m,n）

Rem（取余函数）、div

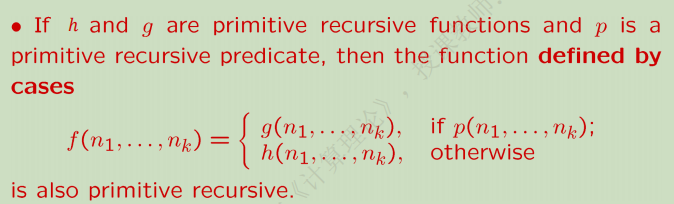
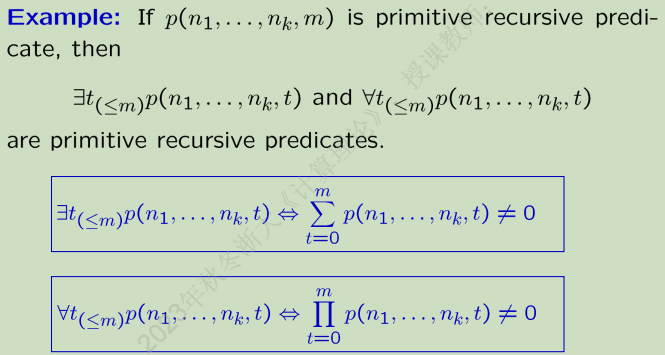
Digit（m,n,p）将n转化为p进制，并输出第m个数字（从后向前数，包含0）

sum function（多函数求和），mult function（多函数累乘）

Negation of any recursive predicate is also a primitive recursive predicate



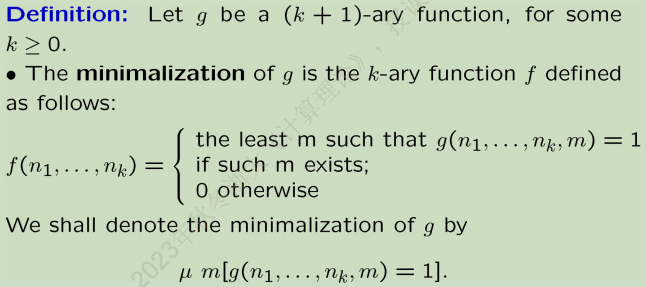
Disjunction and conjunction of two primitive recursive predicates are also primitive recursive predicate

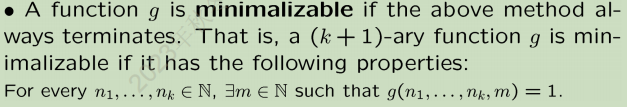
b89cdde8421e931ae22f85f4e28eab8

Y | X表示X is divided exactly by Y（X被Y整除）

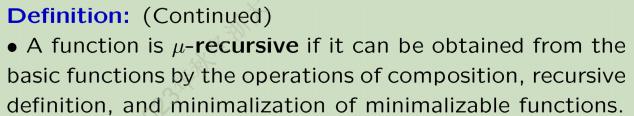
The set of primitive recursive functions is enumerable

Primitive recursive function是recursive function的真子集



Minimalization只能遍历计算m，并不一定会停止

如果实现了minimalization，则可以define与实现logarithm



（填补空白）

Extension of Turing machine（均与一般的Turing machine等价）

Multiple tapes

Two-way infinite tape（左右移动方向均无限，只有一条纸带）

Multiple heads

Mult-dimensional tape

No-determinism

构造Turing machine的组合时，需要引入新的halting状态（用于处理TM跳转失败的情况）

每一个TM semidecide only one language

TM可能不decide any language

Recursive language在union、concat...、intersection、complement、kleen star下均闭包

R.E. 仅对于complement不闭包

Context-sensitive language（上下文相关语言） --- CSG

Linear bounded automata

Context-free只有一个non-terminal symbol

Grammar可以有任意个non-terminal symbol

Algorithm：always halt at all input ---> recursive

Church-Turing thesis：“that can be rendered as Turing machines which halts at all inputs, will be rightfully called as the algorithm”

Turing machine只能解决一个particular problem

Universal Turing Machine（类似于可编程）

将TM，w均转化为字符串，以“TM”“w”表示（encode过程）

“TM”更多的是表示configuration的转变关系

对于symbol而言，额外考虑<-- / -->（blank 00，左起始01，<-- 10，--> 11）以a为起始

对于state而言（0 代表 start state）以q为起始

Conventional TM ---> 仅对映一个语言

Universal TM ---> 类似于programming language

使用三个条带：第一条带记录w的编码，第二条带记录原TM的编码（转换关系），第三条带记录当前的state（用于转变）

每一step需要修改第一条与第三条条带

Halting problem 是 non-recursive language

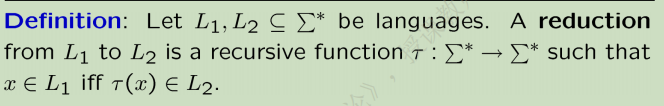
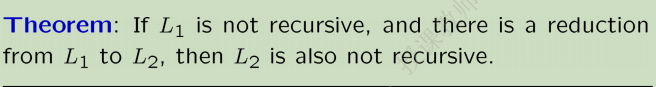
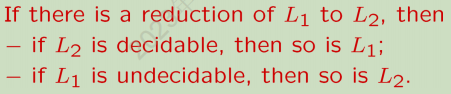
--- recursive enumerable语言

Recursive enumerable语言的complement可以不是r.e. language

Any algorithm can be turned into a TM that halts on all inputs.

若一个problem没有对映的algorithm --- undecidable / unsolvable

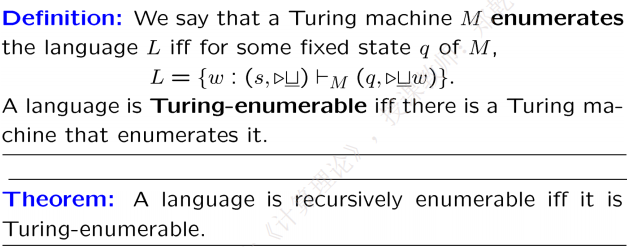
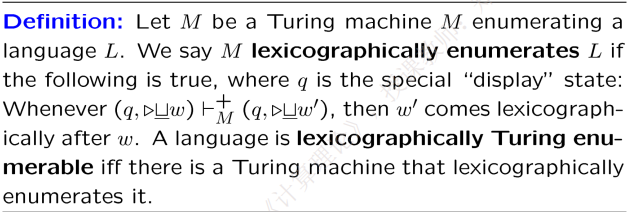
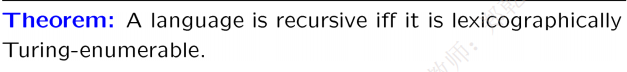
All r.e. language can reduce to halting problem; H is complete for the r.e. language.



Decidable与recursive等价，undecidable与recursive enumerable等价

Undecidable（unsolvable）没有相应的algorithm exist

Context free language右边可以有多个non-terminal symbol和terminal symbol，但是左边只能有一个non-terminal symbol

789d120a10a65d3feb729fabbc287a3