计算理论*

Shane Wang mllover.com

12-26-2013

目录•导航

核心知识点 2012/2011/2010年试卷答案解析

1 预备

一 3个基本的证明技术：数学归纳法、鸽巢原理和对角化方法。对角化原理也可以用在无穷集合上。利用对角化原理可以证明下面的命题。

1 集合2^N是不可数的。

2 区间$[0,1]$上全体实数是不可数的。

二 自反传递闭包：设$R \subseteq A^2$是集合A上的有向图。R的自反传递闭包是关系$R^* = \{(a,b) : a,b \in A$且在R中有从a到b的通路$\}$。

三 函数增长率与前面常数无关。

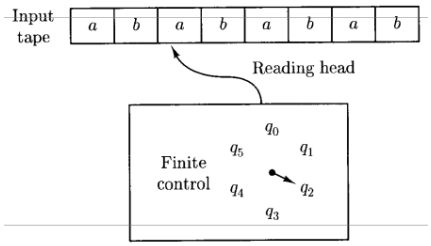
2 定义

- 空串： 用e表示
- 字母： 用希腊字母表示u,v,w,x,y,z
- 字母表Σ: 字母表Σ上所有字符串（包括空串）的集合
- 连接： 连接两个字符串，用$x \circ y$表示。或简写为xy
- 子串：串中的一部分
- 反转：w，记作w^R
- 语言： Σ^*的任一子集，$\Sigma^*, \emptyset$和Σ都是语言。 如果Σ是有穷字母表，Σ^*是可数无穷集合
- 补A: $\Sigma^* - A$
- 语言连接： 设L_1, L_2是语言， 它们的连接定义为 $L = \{w \in \Sigma^* : w = x \circ y$,其中$x \in L_1$,且$y \in L_2\}$
- 语言Kleene Star: L中零个或多个字符串得到的所有字符串的集合 $L^* = \{w \in \Sigma^* : w = w_1 \circ ... \circ w_k$,其中$k \geq 0$且$w_1, ...w_k \in L$
- L^+: LL^*, 可以把L^+看作是L在连接运算下的闭包
- 正则表达式表示： 一般用$\cap, ^*, \emptyset, (,)$来描述
- 正则语言： 所用可以使用正则表达式描述的语言

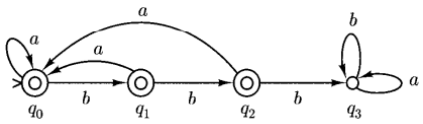
*注意，只对金小刚教授版的计算理论适用

3 DFA

- 确定型有穷自动机:
  $M = (K, \Sigma, \delta, s, F)$,其中 K是用穷自动机的状态集合
  Σ是字母表
  $s \in K$是初始状态集合
  $F \subseteq K$是终结状态集合
  δ是从$K \times \Sigma$到K的函数，叫转移函数
- 注意$\delta(p, a)$是一个状态。
- 用穷自动机格局：机器(有穷控制器、读头及输入带)在相继时刻的状态。不允许确定性有穷自动机回读。它是$K \times \Sigma^*$的任一元素。比如下图表示格局$(q_2, ababab)$



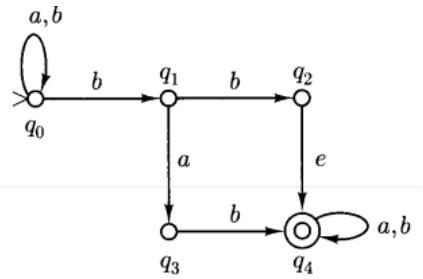
- $\vdash_M$: 一步生成的意思。如果有两个格局$(q, w), (q', w')$,和一个转换函数$\delta(q, a) = q'$，那么我们称(q, w)一步产生(q', w')。$\vdash_M^*$表示零步或或多步产生
- M接受的语言： M接受的所有字符串的集合，记作$L(M)$
- 可以用状态图描述DFA



- 例子$L(M) = \{w \in \{a, b\}^* : w$不含有连续3个b$\}$的DFA。状态图如上图，其中$K, \Sigma, \delta, s, F$分别是：
  $K = \{q_0, q_1, q_2, q_3\}$,
  $\Sigma = \{a, b\}$,
  $s = q_0$,
  $F = \{q_0, q_1, q_2\}$

4 NFA

- 不确定性自动机与确定自动机的区别在于它可以在读入e时转移。它与DFA的定义类似，区别只是转移函数的定义发生了变化：
  $\Delta \subseteq K \times (\Sigma \cup \{e\}) \times K$
- $\vdash_M$: 不一定是函数，因为可以e转移。某些格局可能有若干个（包括零个）有序对(q', w'),使得$(q, w) \vdash_M (q', w')$
- 下面是一个例子：



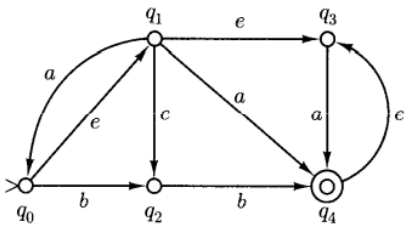
$$
\begin{aligned}
&K = \{q_0, q_1, q_2, q_3, q_4\},\\
&\Sigma = \{a, b\}\\
&s = q_0\\
&F = \{q_4\}\\
&and\\
&\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1),\\
&(q_1, b, q_2), (q_1, a, q_3), (q_2, e, q_4),\\
&(q_3, b, q_4), (q_4, a, q_4), (q_4, b, q_4)\}
\end{aligned}
$$

一个字符串被非确定有穷自动机接受，当且仅当至少有一个引导到终结状态的计算序列。所以上例的$bababab \in L(M)$

- 对于每一台非确定型有穷自动机，有一台等价的确定型有穷自动机
- 非确定型确定型自动机转换为确定型自动机的过程是将非确定型自动机一次可以转移的所有状态的集合看成要转换的确定型自动机的一个状态。构造后的$M^{'}$可以描述为$M^{'} = (K^{'}, \Sigma, \delta^{'}, s^{'}, F^{'})$其中

$$
\begin{aligned}
&K^{'} = 2^K\\
&s^{'} = E(s)\\
&F^{'} = \{Q \subseteq K : Q \cap F \neq \emptyset\}\\
&\delta^{'}(Q, a) = \cup\{E(p) : p \in K, q \in Q, (q, a, p) \in \Delta\}
\end{aligned}
$$

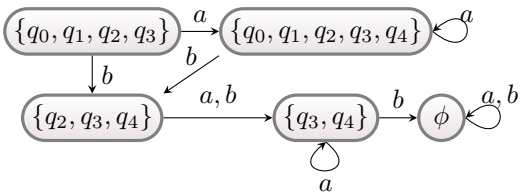
下面例子中，$\delta^{'}(q_1, a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}$。非确型自动机转化为确定型自动机在下图描述。当然，有时最终的状态数目还可以优化，以期达到最小值。



$$
\begin{aligned}
&S^{'} = E(q_0) = \{q_0, q_1, q_2, q_3\}\\
&because \quad (q_1, a, q_0), (q_1, a, q_4), (q_3, a, q_4),\\
&\Rightarrow \delta'(s^{'}, a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}\\
&because \quad (q_0, b, q_2), (q_2, b, q_4)\\
&\Rightarrow \delta'(s^{'}, b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}
\end{aligned}
$$

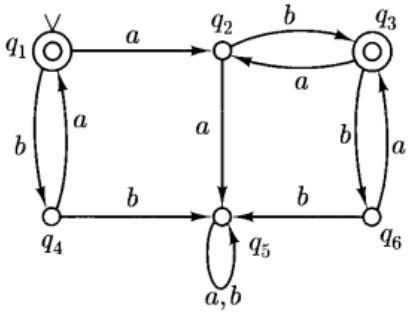
And

$$
\begin{aligned}
&\delta'(\{q_0, q_1, q_2, q_3, q_4\}, a) = \{q_0, q_1, q_2, q_3, q_4\}\\
&\delta'(\{q_0, q_1, q_2, q_3, q_4\}, b) = \{q_2, q_3, q_4\}\\
&\delta'(\{q_2, q_3, q_4\}, a) = E(q_4) = \{q_3, q_4\}\\
&\delta'(\{q_2, q_3, q_4\}, b) = E(q_4) = \{q_3, q_4\}\\
&\delta'(\{q_3, q_4\}, a) = E(q_4) = \{q_3, q_4\}\\
&\delta'(\{q_3, q_4\}, b) = \phi\\
&\delta'(\phi, b) = \phi\\
&\delta'(\phi, b) = \phi
\end{aligned}
$$



5 正则语言和非正则语言

- 能用正则表达式表示或被确定型有穷自动机识别的语言是正则语言。常常用于证明。非正则语言目前还没有强有力的办法证明。 **技巧**：证明正则语言，可以把它用正则语言表示。若能表示，必然是正则的
- 正则语言满足的两条性质
  - 从左到右扫描字符串，确定字符串是否属于该语言所需存储量必须有界。显然$\{a^nb^n : n \geq 0\}$不是正则的
  - 有穷多个字符串的正则语言可以用带圈的有穷自动机和含Kleene Star的正则表达式表示。这样的语言一定有某种简单的重复构造的无穷子集。显然$\{a^n : n \geq 1$是一个素数$\}$不是正则语言，因为它没有简单的周期性
- 泵定理：L是正则语言，则存在长度大于$n \geq 1$的字符串w，则w可以写成 $w = xyz$,其中$y \neq e, |xy| \leq n$, 则对于$i \geq 0, xy^iz \in L$
  泵定理的证明也是显然的。可以通过鸽巢原理来证明:
  假设确定型自动机的状态数目$|w| = n$，而$(q_0, w_1, w_2...w_n) \vdash_M (q_1, w_2...w_n) \vdash_M ... \vdash_M (q_n, e)$有$n+1$个状态。所以必有两个状态相同。记这两个状态为$q_i$和$q_j$，那么，$q_i$到$q_j$的串可以重复任意次$i \geq 0$,满足$xy^iz \in L$ 泵定理常常用来证明是语言不是正则语言
- 语言$L = \{a^ib^i : i \geq 0\}$不是正则的。可以用泵定理来证明：设状态机数目n,那么字符串$w = a^nb^n \in L$，且$|xy| \leq n$。因此y不可能含有b。所以$y = a^r$。但是$xz = a^{n-r}b^n \notin L$
- 语言$L = \{a^n : n$是素数$\}$不是正则的。证明：令$x = a^p, y = a^q, z = a^r$其中$p, r \geq 0$且$q > 0$。根据定理有$xy^nz \in L$(因为$p+nq+r$是素数)。但这是不可能的。令$n = p+2q+r+2 = (q+1)(p+2q+r)$是合数。
- 语言$L = \{w \in \{a, b\}^* : w$中$a$和$b$的个数相同$\}$不是正则的。可以用封闭性证明。因为$L \cap a^*b^*$不正则，所以$L$不正则。
- 等价关系$L \subseteq \Sigma^*$是一个语言$x, y \in \Sigma^*$。对所有$z \in \Sigma^*, xz \in L \quad iff \quad yz \in L$，则x和y等价。$x \approx_L y$。 M相对于确定型自动机等价$(x \sim_M y)$意味着M把x, y从s带入同一个状态。 $(s, x/y) \vdash_M^* (q, e)$
  如果$\sim$是$\approx$的一个细化，那么$\sim$的每个等价类包含在$\approx$。 机器构造等价类举例：



The equivalence class as following:
(1) $E_{q_1} = (ba)^*$
(2) $E_{q_2} = La$
(3) $E_{q_3} = abL$
(4) $E_{q_4} = (ba)^*b$
(5) $E_{q_5} = L(bb \cup aa) \sum^*$
(6) $E_{q_6} = abLb$

- Myhill-Nerode 定理：$L \subseteq \Sigma^*$是一个正则语言，则有一台接受L的确定型自动机，它的状态数恰好与$\approx_L$的等价类一样多
- 推论： 语言L是正则的当且仅当$\approx_L$有有穷个等价类

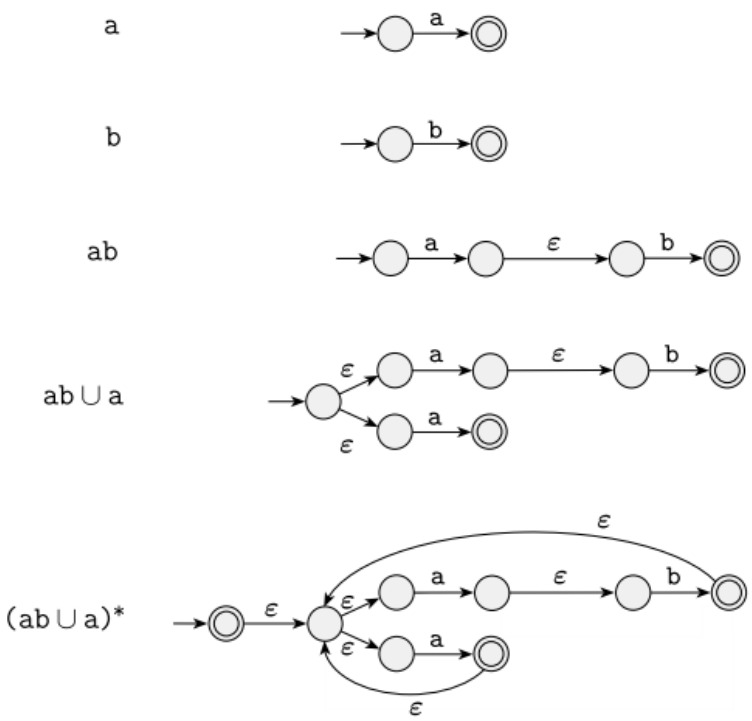
6 考试 之 正则语言和有穷自动机

考试的考点：

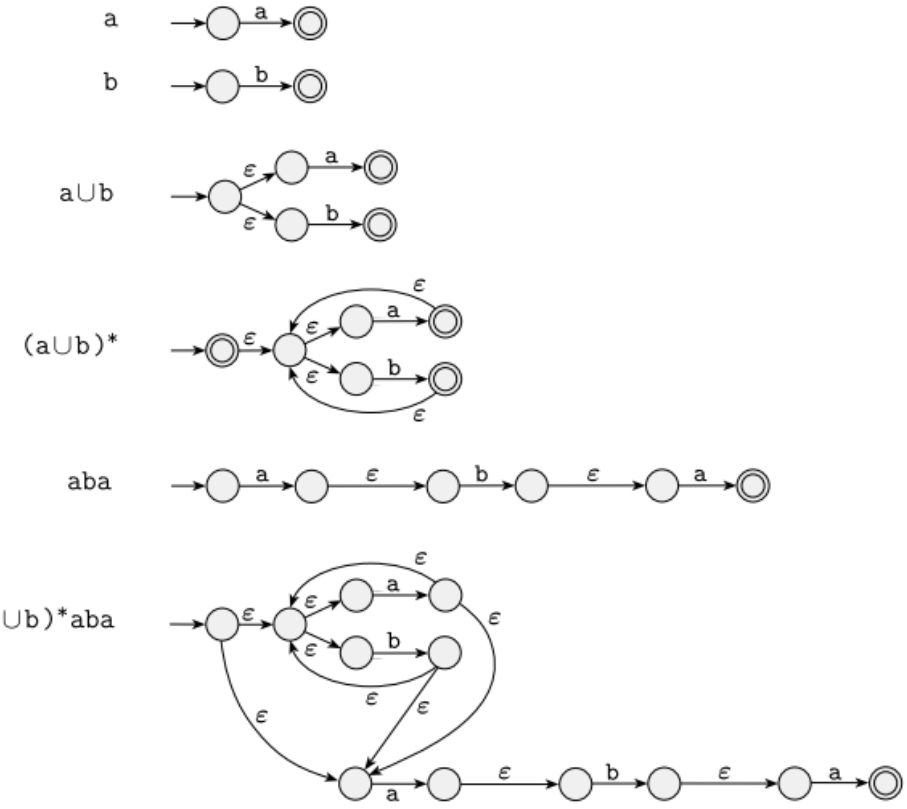
- Regular Expression
- Deterministic finite automata(DFA)
- Non-deterministic finite automata(NFA)
- closure properties and Pumping Theorem
- Minimization(graduated course)

可能考的问题：

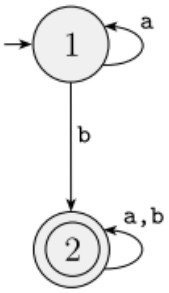
- DFA或者NFA转化为正则语言
- 正则语言转化为DFA或者NFA
- 确定给定语言是否正则。正则可用正则表达式和闭包性质证明。非正则，则用pumping(泵)定理证明
- 下面是上面问题的例子(注意这里ϵ指的是e)
  - 将$(ab \cup a)^*$转换成一台NFA
    解：



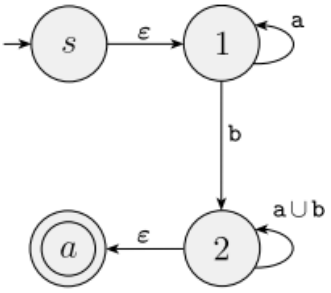
– 将$(a \cup b)^*aba$转换成一台NFA
解：



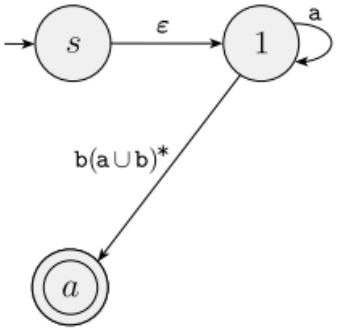
– 将下面DFA(a))转换为正则语言
解：



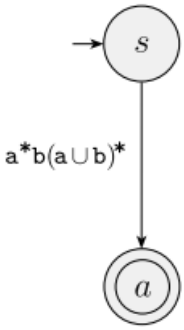
(a)



(b)

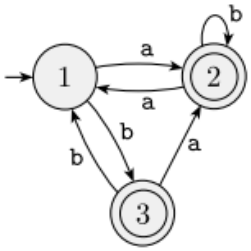


(c)

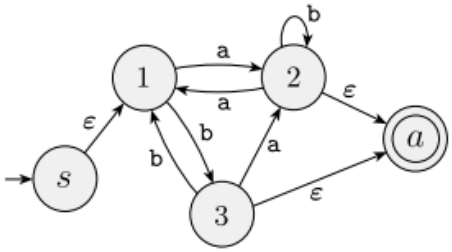


(d)

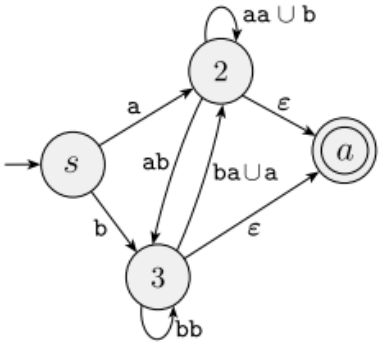
– 将下面(a))转换为正则语言
解：



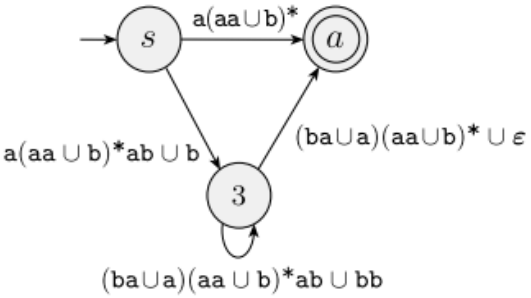
(a)



(b)



(c)



(d)

(a(aa∪b)*ab∪b)((ba∪a)(aa∪b)*ab∪bb)*((ba∪a)(aa∪b)*∪ε)∪a(aa∪b)*

(e)

- 常见非正则语言列表
  - $a^n b^n, n \geq 0$
  - $w|w$中0和1的个数相同
  - $ww|w \in \{0,1\}^*$
  - $ww^R : w \in \{a,b\}^*$
  - 平衡括号(是上下文无关的)
- 正则运算的封闭性：正则语言在有限次并，可数次交，补，连接，Kleene Star，反转，差集下封闭
- 其它性质
  - DFA与NFA等价
  - 语言正则，当且仅当有一台非确定型（确定型）自动机识别它

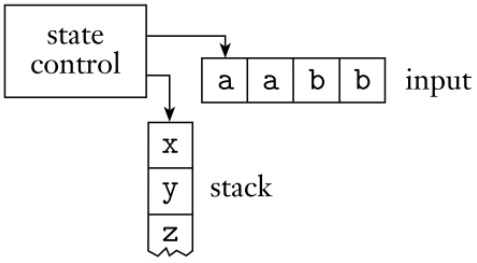
7 上下文无关语言

- 上下文无关文法[1]G是一个四元组(V, Σ, R, S),其中：
  V是字母表，
  Σ是终结符集合，它是V的子集，

[1]注意这样一种隐含的对应关系(正则语言≃正则表达式≃确定型自动机)、(上下文无关语言≃上下文无关文法≃下推自动机). 注意，理论上这种类比是错误的。

R是规则集合，他是$(V-\Sigma)\times V^*$的有穷子集。
$S\in V-\Sigma$是起始符。

- G生成的语言叫上下文无关语言。用$L(G)$表示
- 例子：考虑上下文无关文法$G=(V,\Sigma,R,S)$，其中$V=S,a,b,\Sigma=\{a,b\}$, 而R包括规则$S\to aSb$和$S\to e$可以有推导: $S\Rightarrow aSb\Rightarrow aaSbb\Rightarrow aabb$
- 所有的正则语言都是上下文无关的。
- 上下文无关文法生成的语言w在G中可能有很多推导。推导的过程可以用树表示，叫语法分析树。
- 最左推导：每次只解开最左边的终结符的推导叫最左推导。最右推导类似
- 下述命题等价：
  - $A\xrightarrow{*}w$
  - 有一颗根为A、结果为w的语法分析树
  - 有最左推导到w
  - 有最右推导到w
- 上下文无关文法最左推导或最右推导也可能不唯一。这种现象称为歧义。有些歧义可以通过引入新的非终结符消去，但是有的则不可以。这被称为固有歧义。
- 下推自动机可以识别上下文无关语言。与确定型自动机相比多了一个栈。



下推自动机由六元组$M=(K,\Sigma,\Gamma,\Delta,s,F)$表示，其中
K是有穷的状态集合
Σ是一个字母表所有输入符号
Γ是字母表的栈符号
$s\in K$是初始状态
$F\subseteq K$是终结状态集合
Δ是转移关系，它是$(K\times(\Sigma\cup\{e\})\times\Gamma^*)\times(K\times\Gamma^*)$
这个定义是非确定的。以后还要给出确定型下推自动机

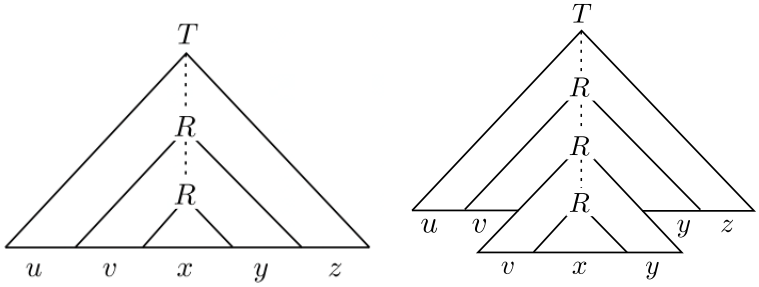
- 直观上讲，M的转移$((p,a,\beta),(q,\gamma))\in\Delta$的含义是状态$p$读到$a$,在栈顶用$\gamma$代替$\beta$, 然后进入状态$q$. 其中$\beta=e$表示不访问输入带
  推入a: $((p,u,e),(q,a))$
  托出a:$((p,u,a),(q,e))$
  格局:$K\times\Sigma^*\times\Gamma^*$. 第一个变量是机器状态；第二个分量是没有读的输入部分；第三个分量是下推存储器的内容 (靠左边的表示在栈顶)
- 语言$L=\{wcw^R:w\in\{a,b\}^*\}$设计的PDA包含的五个转移：
  (1) $((s,a,e),(s,a))$
  (2) $((s,b,e),(s,b))$
  (3) $((s,c,e),(f,e))$
  (4) $((f,a,a),(f,e))$
  (5) $((f,b,b),(f,e))$
- 语言$L=\{w\in\{a,b\}^*:w$中a与b的个数相同$\}$:
  这里$K=\{s,q,f\},\Sigma=\{a,b\},\Gamma=\{a,b,c\},F=\{f\}$, 而$\Delta$列表如下：

(1) $((s,e,e),(q,c))$
(2) $((q,a,c),(q,ac))$
(3) $((q,a,a),(q,aa))$
(4) $((q,a,b),(q,e))$
(5) $((q,b,c),(q,bc))$
(6) $((q,b,b),(q,bb))$
(7) $((q,b,a),(q,e))$
(8) $((q,e,e),(f,e))$

这里的直观解释就是：当读到和栈顶相异的字符时，消去栈顶元素。否则保存到栈顶元素

- 任一台有穷自动机都可以看作没有对栈进行任何操作的下推自动机
- 下推自动机和上下文无关文法：
  定理：下推自动机接受的语言正好是上下文无关语言类
  引理：每一个上下文无关语言都被一台下推自动机接受
  引理：如果一个语言被一台下推自动机接受，则它是上下文无关语言

- 上下文无关文法⇒ PDA的一般过程举例：
  假设有下面的文法
  $G=(V,\Sigma,R,S)$,其中
  $V=\{S,a,b,c\}$
  $\Sigma=\{a,b,c\}$
  $R=\{S\to aSa,S\to bSb,S\to c\}$
  它生成的语言$\{wcw^R:w\in\{a,b\}^*$可以如此构造：
  $\Delta=$
  $\{((p,e,e),(q,S)),$
  $((q,e,S),(q,aSa)),$
  $((q,e,S),(q,bSb)),$
  $((q,e,S),(q,c)),$
  $((q,a,a),(q,e)),$
  $((q,b,b),(q,e)),$
  $((q,c,c),(q,e))\}$
- 上下文无关语言的封闭性(上下文无关语言与非上下文无关语言)：
  上下文无关语言在并、连接和Kleene Star下是封闭的
  上下文无关语言与一个正则语言的交是上下文无关语言
  上下文无关语言在交、补下不封闭
  确定型上下文无关语言在补运算下封闭
  上下文无关语言反转(reverse)封闭
- 泵定理
  泵定理常被用来反证一个语言不是上下文无关语言
  引理：G的任意一棵高度为h的语法分析树的结果的长度不超过$\phi(G)^h$
  泵定理：设$G=(V,\Sigma,R,S)$是一个上下文无关文法。那么$L(G)$中任一长度大于$\phi(G)^{|V-\Sigma|}$的字符串w可以写成$w=uvxyz$使得v或y不是空串，并且对每一个$n\geq 0$,$uv^nxy^nz\in L(G)$,其中$\phi(G)$是规则右边最大的符号数目。下图很美妙地说明了这一点。



- 语言$L=\{a^nb^nc^n:n\geq 0\}$不是上下文无关的。证明也是比较直观的。假设n大于所要求的最要长度$\frac{1}{3}\phi(G)^{|V-\Sigma|}$,那么只需要证明$uv^ixy^iz\notin L(G)$。显然，不论怎么假设，上式都不在属于$L(G)$。因为如果$v,y$包含$a,b,c$,$v,y$其中一个至少含有$a,b,c$的两个字符，于是$uv^2xy^2z$不再平衡。如果$v,y$含有一个或两个$a,b,c$中的元素，那么$a,b$也不会平衡。故语言不是上下文无关语言
- 语言$L=\{a^n:n$是个素数$\}$不是上下文无关语言。证明是：假设有足够大的素数满足大于$\phi(G)^{|V-\Sigma|}$的条件。于是w可以写成$uvxyz$。令vy长度为q,uxz长度为r,那么，根据泵定理可得，$n\times q+r$是素数。但是上式有可能不是素数。所以它不是上下文无关语言
- 语言$L=\{w\in\{a,b,c\}^*:w$中a,b,c的个数相等$\}$不是上下文无关的。这个需要用到上面提到的“上下文无关语言与正则语言的交是上下文无关的语言”定理。由于她和正则语言$a^*b^*c^*$的交是$a^nb^nc^n$。所以它不是上下文无关语言
- 注意$\{a^nb^nc^m:m,n\geq 0\}$,$\{a^mb^nc^n:m,n\geq 0\}$是上下文无关的
- 定理：
  - 有多项式算法，任一个上下文无关文法，构造一台等价的下推自动机
  - 有多项式算法，任一台下推自动机，构造一个等价的上下文无关文法
  - 有多项式算法，任给一个上下文无关文法G和一个字符串x,判断是否$x\in L(G)$
- 确定型上下文无关语言：如果下推自动机M的计算中每一个格局至多能够有一个格局接在它后面，则称M是确定型的。

8 考试 之 上下文无关文法

考试的考点：

- context-free grammar
- pushdown automata
- closure properties and pumping theorem

可能考的问题：

- context-free language $\Rightarrow$ context-free grammar
- context-free language $\Rightarrow$ PDA
- content-free grammar $\Rightarrow$ PDA
- 给定语言是否是上下文无关语言(闭包和pump定理)
- 下面是一些例子
  - $L=\{a^ib^jc^k|0\leq i\leq j\leq k\}$不是上下文无关的

9 Turing机

- Turing机是五元组(K,Σ,δ,s,H),其中
  K是状态有穷集
  Σ是字母表，包含空格符$\sqcup$和左端符$\triangleright$ $s\in K$是初始状态
  $H\subseteq K$是停机状态
  δ是转移函数，它是从$(K-H)\times\Sigma$到$K\times(\Sigma\cup\{\leftarrow,\rightarrow\})$的函数，并且满足：
  (a) 对所有$q\in K-H$,若$\delta(q,\triangleright)=(p,b)$则$b=\rightarrow$
  (b) 对所有$q\in K-H$和$a\in\Sigma$,若$\delta(q,a)=(p,b)$则$b\neq\triangleright$
- 例子：考虑下面的图灵机$M=(K,\Sigma,\delta,s,\{h\})$,其中$K=\{q_0,q_1,h\},\Sigma=\{a,\sqcup,\triangleright\},s=q_0,\delta$由下图给出(12页最下方),其中有些行是不必要的，但是为了函数定义的完整性，这里都给出了

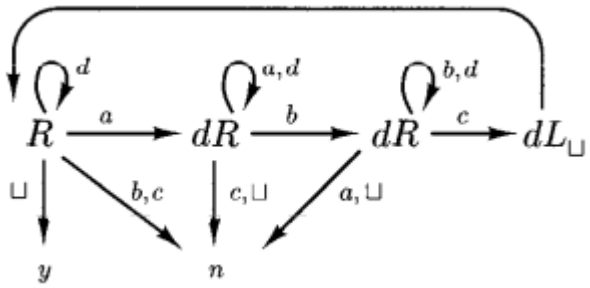
- 格局：图灵机的格局是$K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{e\})$,所以可以看出，$(q, \triangleright baa, abc\sqcup)$和$(q, \sqcup aa, ba)$不是格局。格局也可以用简化的下划线表示如$\triangleright\underline{a}aba$
  格局转换例子：

  - $\delta(q_1, a) = (q_2, b)$: $(q_1, w\underline{a}u) \vdash_M (q_2, w\underline{b}u)$
  - $\delta(q_1, a) = (q_2, \leftarrow)$: $(q_1, wb\underline{a}u) \vdash_M (q_2, w\underline{b}au)$,$(q_1, wb\underline{\sqcup}) \vdash_M (q_2, w\underline{b})$
  - $\delta(q_1, a) = (q_2, \rightarrow)$: $(q_1, w\underline{a}bu) \vdash_M (q_2, wa\underline{b}u)$,$(q_1, w\underline{a}) \vdash_M (q_2, wa\underline{\sqcup})$

- 例子：上面提到的例子对$q_1, \underline{\sqcup}aaaa)$产生的格局是：

$$
\begin{aligned}
(q_1, \triangleright\underline{\sqcup}aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) \\
&\vdash_M (q_1, \triangleright \sqcup \underline{\sqcup}aaa) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup\underline{a}aa) \\
&\vdash_M (q_1, \triangleright \sqcup \sqcup\underline{\sqcup}aa) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) \\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \underline{\sqcup}a) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup\underline{a}) \\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup\underline{\sqcup}) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup\underline{\sqcup}) \\
&\vdash_M (h, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup\underline{\sqcup})
\end{aligned}
$$

- 为了描述方便，有时候会用$R_\sqcup, L_\sqcup, R_\square, L_\square$分别表示：寻找到右方第一个空格，左方第一个空格，右方第一个非空格，左方第一个非空格
- 图灵机的表示： 最左端$\triangleright$，再留一个空格符$\sqcup$接着就是输入字符串w。跟着全是空格
- 机器**判定**语言L：若$w \in L$则M接受w；若$w \notin L$则M拒绝w,那么我们说M判定语言L。若存在图灵机判定语言L则L称为递归的
- 语言$L = \{a^n b^n c^n : n \geq 0\}$的图灵机的状态图如下所示



- 设$M = (K, \Sigma, \delta, s, H)$是Turing机。如果$w \in L$当且仅当$M$在输入$w$上停机，则我们说$M$半判定$L$。语言是递归可枚举的当且仅当存在Turing机$M$半判定$L$。不过，若$w \in \Sigma_0^* - L$，则$M$必然用不进入停机状态或者不会停止在接受状态。因为任何非停机格局都产生某种其他格局(δ是全函数)，所以在这种情形里机器必然无限地继续计算。半判定语言的图灵机都不是算法。
- 若语言是递归的，则它时递归可枚举的。但是存在非递归的递归可枚举语言

q_0	a	$(q1, \sqcup)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	a	$(q0, a)$
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$

- 若 L是递归的，则它的补$\overline{L}$也是递归的
- 下面是**数值函数**部分。数值函数中，基本函数包括三类：
  (a) k**元零函数**： $k \geq 0$,对所有$n_1, ..., n_k \in N, zero_k(n_1, ..., n_k) = 0$
  (b) $k \geq j > 0$,第j个k 元**恒等函数**定义为对所有$n_1, ..., n_k \in N, id_{k,j}(n_1, ..., n_k) = n_j$
  (c) $n \in N, succ(n) = n + 1$定义为**后继函数**
- 函数的两种基本组合方式：
  - **合成**(composition of g):如果g是k元函数，h是l元函数，那么$f(n_1, ..., n_l) = g(h_1(n_1, ..., n_l), ..., h_k(n_1, ..., n_l))$是$l$元函数
  - g和h**递归**定义的函数(function defined recursively by g and h)：g是k元函数，h是$k+2$元函数，则
    $f(n_1, ..., n_k, 0) = g(n_1, ..., n_k)$
    $f(n_1, ..., n_k, m+1) = h(n_1, ..., n_k, m, f(n_1, ..., n_k, m))$是$k+1$元函数

  原始递归函数(primitive recursive functions)是有基本函数和用基本函数通过任意次相继应用合成和递归定义所获得的函数
- 举例： $plus2(n) = n + 2$是原始递归的。令$h = g = succ(n)$,那么，$g(h) = succ(succ(n)) = n + 1 + 1 = n + 2$。 乘法函数$mult(m, n) = m \cdot n$被递归地定义为$mult(m, 0) = zero(m), mult(m, n + 1) = plus(m, mult(m, n))$。指数函数$exp(m, n) = m^n$被定义为$exp(m, 0) = succ(zero(m)), exp(m, n + 1) = mult(m, exp(m, n))$
  常值函数是原始递归的。$sgn(n)$也是原始递归的。因为它可以描述成常值函数的形式
  因此，循着这条路子，$m \cdot (n + m)^2 + 178^m$。$maxm - n, 0, div(m, n), rem(m, n)$
- 原始递归谓词是只取0和1的原始递归函数
- 分段函数

$$f(n_1, ..., n_k) = \begin{cases} g(n_1, ..., n_k) & if \quad p(n_1, ..., n_k) \\ h(n_1, ..., n_k) & elsewise \end{cases} \tag{1}$$

  也是原始递归的，因为它可以写成$f(n_1, ..., n_k) = p(n_1, ..., n_k) \cdot g(n_1, ..., n_k) + (1 \sim p(n_1, ..., n_k)) \cdot h(n_1, ..., n_k)$
- 实际上很难设计出不是原始递归的函数☺。原始递归函数总是可以用停机的图灵机计算，而递归函数需要图灵完全系统。
- g的 **极小化**是如下定义的k元函数

$$f(n_1, ..., n_k) = \begin{cases} min(m), & g(n_1, ..., n_k, m) = 1, \quad \exists m \\ 0, & elsewise \end{cases} \tag{2}$$

- 若函数可以从基本函数出发，通过合成，递归定义以及可极小化函数的极小化等操作来获取，则称它为μ递归的。函数是μ递归的，当且仅当它是递归的

10 考试 之 图灵机

考试的考点：

- turing machine
- grammar
- numerical functions
- 了解basic functions, composition, function defined recursively, primitive recursive functions, primitive

可能考的问题：

- 对于一个函数或语言设计一个图灵机
- 给定一个图灵奖，推出它的函数
- 证明一个函数是原始基函数

11 不可判定性

- 在了解不可判定性问题前，要了解一些常见的可判定问题：
  - 检验DFA是否接受给定串的问题
  - 检验NFA是否接受给定串的问题
  - 检验正则表达式是否接受给定串的问题
  - 检验DFA根本不接受任何串的问题
  - 检验两个DFA是否识别同一个语言的问题
  - 检验CFG是否派生特定串的问题
  - 检验CFG根本不派生任何串的问题
  - 每个上下文无关语言都是可判定的语言,但检验上下文无关文法是否派生所有可能的串是不可判定的

  注意:可能考判断题

- 停机问题
  假设$halts(P, X)$表示P函数在输入参数X上是否停机。那么下面问题问题是否停机：
  $diagonal(X)$
  a: if $halts(X, X)$ then goto a else $halt$
  对于这样一段程序，如果我们以$diagonal$和$diagonal$作为参数输入$halts$函数中，我们会得到一个矛盾：如果$diagonal$最终停机了，即$diagonal$判定$diagonal$。这表明$halts(X, X)$返回的是否。但是如果$halts(X, X)$返回否，意味着$diagonal$无法判定输入$diagonal$。这与前面矛盾。

- 递归语言的补是递归的，递归可枚举语言的补不是递归可枚举的

- 递归可枚举和递归语言，及其如何判定可以参考这篇文章：fondle me

- 如果L_1不是递归的，并且存在从L_1到L_2的归约，则L_2也不是递归的。这个定理可以证明以下定理[重要]：
  - 给定Turing机M和输入字符串w,M是否在输入w上停机?
  - 给定Turing机M,M是否在空带上停机?
  - 给定Turing机M,究竟有没有M在上面停机的字符串?
  - 给定Turing机M,M是否在每一个输入字符串上都停机?
  - 给定两台Turing机M_1, M_2,它们是否在同样字符串上停机?
  - 给定Turing机M,M半判定的语言是否正则？是否上下文无关？是否递归?
  - 给定固定Turing机M_0,给定w,M_0是否在w上停机?

  注意，在不可判定性那道大题里，很有可能使用上面提到某个结论。我们要知道，证明L的不可判定性问题往往需要用已知的不可判定性问题来归约。以后的例子中会给出这种证明过程。还要注意的是，上面的**给定字符串**w,与**存在字符串**是不一样的。前者是任意的，后者是有约束的。或者是可枚举的。我们将在11年的大题里看到这一点

- 注意，图灵可识别，可判定，不可判定，半判定，递归，递归可枚举，非递归可枚举之间的关系。图灵可识别即**只需**存在图灵机，对语言中每个字符串都接受，语言外的字符串机器拒绝或不停机。可以看出图灵可识别的语言等价于递归可枚举语言，等价于半判定语言。可判定语言指的是，无论字符串是否属于语言，图灵机都会给出准确的回答(是或否)，不可能陷入无限循环状态。图灵可判定的语言等价于递归语言。图灵不可判定语言包括两个层次：图灵可识别但不是可判定的，图灵不可识别的。可以看出，半判定语言一定属于不可判定语言，但不能等价。因为的确存在图灵不可识别的语言。比如停机问题的补问题就是图灵不可识别的。它当然是图灵不可判定的。也不是半判定的。

- 语言是递归的当且仅当它和它的补都是递归可枚举的

- 语言可判定的当且仅当它和它的补都是图灵可识别的

- 递归语言和递归可枚举语言的封闭性
  递归语言在并，交，补，连接，Kleene Star下封闭(全部封闭)
  递归可枚举语言仅在补下不封闭，其它运算封闭。注意在差集下也不封闭。只有正则和递归差封闭。

- 图灵机M枚举语言L。当且仅当对M的某个固定状态q
  $L = \{w : (s, \triangleright\sqcup) \vdash_M^* (q, \triangleright\sqcup w)\}$.语言是Turing可枚举的当且仅当存在枚举它的Turing机。语言是递归可枚举的当且仅当它是Turing可枚举的证明题

- M以字典序枚举L指每当$(q, \triangleright\sqcup w \vdash_M^+ (q, \triangleright\sqcup w')$时，$w'$就在字典序下排在$w$之后。语言是以字典序枚举Turing可枚举的当且仅当存在以字典序枚举它的Turing机
  语言是递归的当且仅当它是以字典序Turing可枚举

12 考试 之 不可判定性

考试的考点：

- Church-Turing Thesis
- Chomsky hierarchy
- Universal Turing Machine
- Halting Problem
- Some Undecidable problems
- Reduction
- Turing-enumerable and lexicographically Turing enumerable
- Chomsky hierarchy

可能考的问题：

- 证明给定语言是递归的、递归可枚举的还是不可判定的
- 例子·技巧：
  该类证明强烈依赖两种证明技术：归约和反证[2]。当A归约到B时，A不可能比解B更困难。因此，如果B是可判定的，那么A是可判定的。A是不可判定的，那么B一定是不可判定的。上面的一些不可判定的定理很有用。现在来证明上面的部分定理，也顺便展示证明该类问题的普遍技术。现在我们证明检验给定图灵机M,是否在空带上停机是不可判定的：假设该命题成立。那么有图灵机R可以检验这个语言。我们的目标是要根据这个假设再设计一个判定器S,它可以判定停机问题是可判定问题。于是得出结论：假设有误，即以上语言是不可判定的。现在我们focus的问题是如何设计这个判定器S。我们现在已知，如果R接受w,意味着w是空串，那么M不接受w。如果R拒绝w,意味着w不是空串，但此时还不能说M接受w，因为我们得出的结论仅仅是w不是空串，而不是M能接受w。于是为了得到我们期望得到的效果。我们加上这样一条约束：如果输入字符串$\neq w$,令R接受。那样我们只需要构造另一台图灵机M_0:

  (1) 如果$x \neq w$,则拒绝
  (2) 如果$x = w$,则在x上运行M。如果M接受则接受。

  于是我们可以构造如下图灵机：S=

  (1) 用M和w的描述构造图灵机M_0
  (2) 在输入M_0上运行R
  (3) 如果R接受，则拒绝；如果R拒绝，则接受

  构造完后，我们检验：

  - 如果输入字符串为e, 那么S拒绝
  - 如果输入字符串为$x \neq w$, 那么S拒绝

[2]还可以利用计算历史的归约，考试并不要求

– 如果输入字符串为$x = w$, 那么S接受

我们发现S的确是判定器，无论何种情况都可以检验图灵机M对输入字符串w是否停机(或者是否可接受),这也意味着S可以判定停机问题。于是得出矛盾。于是假设是不成立的。即原始问题检验给定图灵机M,是否在空带上停机是不可判定的
当然，考试的证明没有这么原子化，这些定理都可以直接拿来用。不过证明的技术是类似的。证明时还要利用闭包性质和图灵枚举的一些性质，这里不再啰嗦

13 计算复杂性、NP完全性

- 如果语言存在判定它的多项式界限的Turing机，则语言称为多项式可判定语言。记为P。更精确的定义是：P是确定型单带图灵机在多项式时间内可判定的语言类，即：
$P = \bigcup_k TIME(n^k)$
- 常见的P类问题有：
  - 图G中是两点u, v是否有有向路径的问题$(PATH)$
  - x与y互素的问题$(RELPRIME)$
  - 上下文无关语言
- $E = (\text{"}M\text{""}w\text{"} : M$在最多$2^{|w|}$步之后接受"$w$".则$E \notin P$
- 如果存在多项式可计算函数，对于每个$x \in \Sigma^*$下列关系成立:$x \in L_1$当且仅当τ称为从L_1到L_2的多项式规约。若A可以规约到B,如果B拥有多项式时间内的算法，那么A也拥有多项式时间内的算法。即B至少像A一样难
- 如果有语言在多项式时间内可以验证，则称语言为NP问题。一个语言在NP中，当且仅当它能被某个非确定型多项式时间图灵机判定。NP类问题有：
  - 哈密尔顿回路问题
  - 寻找子团问题$CLIQUE$
  - 寻找集合子集之和等于给定值得问题$SUBSET - SUM$
- NP完全性指的是：若B是NP完全的，那么B属于NP并且NP中每个A都多项式时间归约到B
- 为了理解这个定义，我们要知道下面概念：
  - SAT: 我们记$bool$公式为ϕ。$bool$公式的可满足性指的是，给$bool$变量赋值，使得$bool$公式的值为1。可满足性问题就是判定一个$bool$公式是否是可满足的。令$SAT = \{< \phi > | \phi$可满足$bool$公式$\}$
  - 库克-列文定理：$SAT \in P$,当且仅当$P = NP$
  - $A \leq_P B$ 指的是A可以多项式时间归约到B。若$B \in P$,那么$A \in P$
  - NP完全问题是可复合的。即若B是NP完全的，且$B \leq_P C$，C属于NP,则C是NP完全的
  - 合取范式：简而言之，就是析取子式的合取。可以参见wiki。文字(literal)指的是$bool$变量或$bool$变量的非。子句指的是由$\vee$连接起来的若干文字。若子句只有3个文字，则称为$3cnf$
  - $3SAT = \{< \phi > | \phi$可满足的$3cnf$公式$\}$。 $3SAT$可多项式时间归约到$CLIQUE$
  - SAT和$3SAT$都是NP完全的
- **于是：**
为了证明语言是NP的，我们只需证明它是多项式内可验证的。我们可以使用验证机做到这一点。
为了证明语言是NP完全的，我们只需证明它是NP的，并且可以由SAT或$3SAT$规约得到
这是最后一道答题的核心证明技术。没有之一
- 如何证明一个问题可以由SAT或$3SAT$规约得到？构造归约!

14 考试 之 计算复杂性、NP完全性

考试的考点：

- P Problems
- NP Problems
- NP Completeness

可能考的问题：

- 指出一个问题是P问题还是NP问题
- 证明语言是NP问题，和NP完全性

15 试题答案· 分析.Shane Editon [3]

试卷详见附录。含2012，2011，2010试卷。

15.1 DFA,NFA,Regular

- 2012年第2题：
  (1) L_1是正则的.
  因为$L_1 = \{w \in \{a,b\}^* | \#w(a) \cdot \#w(b) \text{ is odd}\}$等价于$L_4 \cap L_5$其中，$L_4$ 是w 中有奇数个 a, L_5 是 w 中有奇数个b
  (2) L_2是不是正则的.
  设状态数目为n. 令$T = xyz$,其中，$|xy| = n$,那么，y必由a组成。那么，$a^{n-1}y^2z = a^{n+1}(bc)^n$.
- 2011年第2题：
  (a) L_1是正则语言.
  因为L_1等价于$L_1 = \{w : \text{w中有偶数个a}\}$。那当然是正则语言。
  (b) L_1不是正则语言
  设状态机状态的数目为n. 设某一子串：$L_0 = x^{'}acy \in L_2$,其中$\#x^{'}(a) + 1 = \#y(a) = n$.
  那么，$pq^3r = x^{'}a^3cy = x^{'}aaacy \notin L_2$
- 2010年第2题：
  (a) 是正则语言。
  设$L_e ven = (00 \cup 01 \cup 10 \cup 11)^*$. 则，$H(L) = L \cap L_e ven$.正则语言的交是正则的。所以$H(L)$是正则的。
  (b) 是上下文无关语言。
  利用正则语言与上下文无关语言的交是正则的，可以得到答案。
- 2010年第3题：
  证明：令$L_1' = acbczz^R = acbcABBA$.设状态机状态数目为$n$.其中，$A$中全是$a$,$B$中全是$b$,且$|acbcA| = n$. 那么，$acbcA'a^iBBA \notin L_1$.

15.2 CFG,PDA

- 2012年第3题：
  (a)

$$
\begin{aligned}
&V = \{a, b, S, A, B\} \\
&\Sigma = \{a, b\} \\
&R = \{S \to AB, \\
&\quad A \to aAa|bAb|cB, \\
&\quad B \to aB|bB|e\}
\end{aligned}
$$

[3] 非官方，仅参考

(b) $K = \{p, q\}, \Gamma = \{a, b, A, B\}, s = p, F = q$

$$
\begin{aligned}
&(p, e, e), (q, AB)\\
&(q, e, A), (q, aAa)\\
&(q, e, A), (q, bAb)\\
&(q, e, A), (q, cB)\\
&(q, e, B), (q, aB)\\
&(q, e, B), (q, bB)\\
&(q, e, B), (q, e)\\
&(q, a, a), (q, e)\\
&(q, b, b), (q, e)
\end{aligned}
$$

- 2011年第3题：

(a)

$$
\begin{aligned}
&V = \{a, b, S, A\}\\
&\Sigma = \{a, b\}\\
&R = \{S \to aAb|bAa\\
&\quad A \to aAa|bAb|e\}
\end{aligned}
$$

(b) $K = \{p, q\}, \Gamma = \{a, b, A, S\}, s = p, F = q$

$$
\begin{aligned}
&(p, e, e), (q, S)\\
&(q, e, S), (q, aAb)\\
&(q, e, S), (q, bAa)\\
&(q, e, A), (q, aAa)\\
&(q, e, A), (q, bAb)\\
&(q, e, A), (q, e)\\
&(q, a, a), (q, e)\\
&(q, b, b), (q, b)
\end{aligned}
$$

- 2010年第3题：

(a)

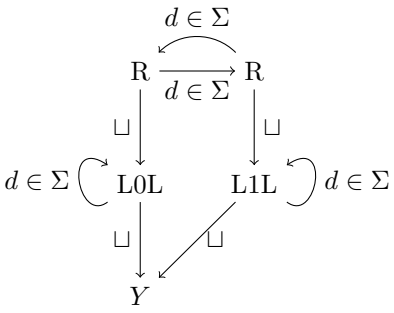
$$
\begin{aligned}
&V = \{a, b, S, A, B, C\}\\
&\Sigma = \{a, b\}\\
&R = \{S \to BaAb|aAbC\\
&\quad A \to aAb|e\\
&\quad B \to Ba|a\\
&\quad C \to bC|b\}
\end{aligned}
$$

(b) $K = \{p, q\}, \Gamma = \{a, b, A, B, C, S\}, s = p, F = q$

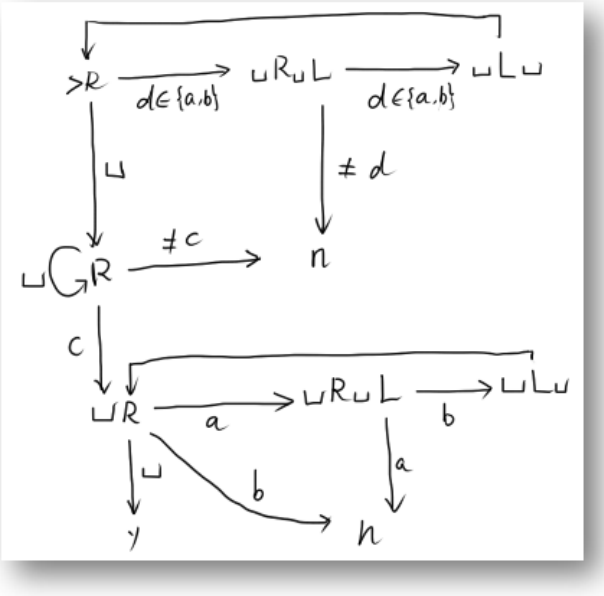
$$
\begin{aligned}
&(p, e, e), (q, S)\\
&(q, e, S), (q, BaAb)\\
&(q, e, S), (q, aAbC)\\
&(q, e, A), (q, aAb)\\
&(q, e, A), (q, e)\\
&(q, e, B), (q, Ba)\\
&(q, e, B), (q, a)\\
&(q, e, C), (q, bC)\\
&(q, e, C), (q, b)\\
&(q, a, a), (q, e)\\
&(q, b, b), (q, e)
\end{aligned}
$$

15.3 TM

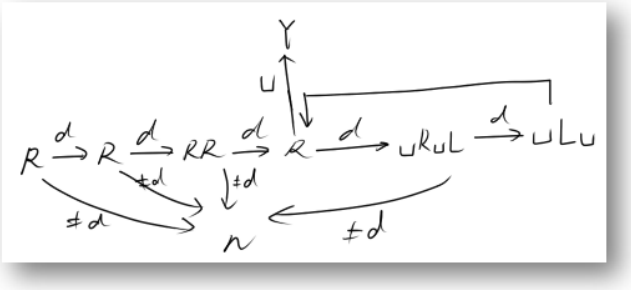
- 2012年第4题：



- 2011年第4题：



- 2010年第4题：



15.4 Primitive Recurisive Functions

- 2012年无此类大题。

- 2011年第5题：
  设：Composit(x) $= \exists_{<x} m \exists_{<x} n((m \neq 1) \wedge (n \neq 1) \wedge (x = m \cdot n))$

那么：
原式$=(x+1)^y \times Composit(x) \times Composit(y)+0$
故是原始递归函数。

- 2010年无此类大题。

15.5 Undecidability

- 2012年第5题：
  (a) 枚举无限可数个回文，至少有一个使得M接受。其它形式广义图灵机U继续运行。故是递归可枚举的。
  (b) 反证法。L_2 是L_1的补。若L_2是递归可枚举的。那么，L_1是递归的。矛盾。故L_2是递归可枚举的。
- 2011年第6题：
  (a) 与上面证明方法类似。
  (b) 与上面证明方法类似。
- 2010年第6题：
  (a) 与上面证明方法类似。
  (b) 与上面证明方法类似。

15.6 P,NP,NP-Complete

- 2012年第6题：
  (a) 在多项式时间内被验证的问题。或者被非确定型图灵机多项式内判定的问题。
  (b) 对于输入$\phi(x_1,...,x_n)$, NTM非确定地猜测两个值是否使得ϕ满足。所以$DOUBLE-SAT \in NP$
  (c) 令$\phi'(x_1,...,x_n,y)=\phi(x_1,...,x_n) \wedge (y \vee \overline{y})$.
  于是，当$\phi(x_1,...,x_n) \in SAT \Rightarrow \phi'(x_1,..,x_n,y) \in DOUBLE-SAT$
  $\phi(x_1,...,x_n) \notin SAT \Rightarrow \phi'(x_1,..,x_n,y) \notin DOUBLE-SAT$
  所以，$SAT \leq_P DOUBLE-SAT$. 所以$DOUBLE-SAT$ 是 $NP-Complete$.
- 2011年第7题：
  (a) 证明：给定赋值，可以在多项式时间内逐一验证是否每个clause中，一个文字为真一个为假,且多项式满足。
  (b) 证明：令3SAT中，$\phi_i=(x_1 \vee x_2 \vee x_3)$. 构造$\alpha_i=(x_1 \vee x_2 \vee z_i) \wedge (\neg z_i \vee x_3 \vee false)$. 易知：
  $\phi \in 3SAT \Rightarrow \alpha \in \neq SAT$
  $\alpha \in \neq SAT \Rightarrow \phi \in 3SAT$
  所以：
  $\alpha \in \neq SAT \Leftrightarrow \phi \in 3SAT$
  所以原问题时$NP-Complete$.