«نظریه زبان‌ها و ماشین‌ها»

Automata Theory

① deals with the defintions and properties of mathematical models of computation

با مدل های ریاضی برای محاسبات / معرفی مدل ها ، قابلیت ها و محدودیت های آن ها

② these models play a role in several applied areas of computer science.

یکی از این مدل ها • finite automation کاربرد → text processing, compilers, hardware design

کاربردی

یکی دیگر از مدل ها • context-free grammer کاربرد → programming language, AI (artificial intelligence)

این مدل ها به نوعی بیان کننده کامپیوتر هستند
یک تقریب از واقعیت

Computability Theory

تئوری محاسبه پذیری

هدف ، تقسیم بندی مسائل به مسائل حل پذیر به شکل الگوریتمی و مسائل حل ناپذیر به شکل الگوریتمی

in this theory, the classification of problem is by those that are solvable and those that are not. * computability theory introduces several of the concepts used in complexity theory.

Complexity Theory

دسته‌بندی مسائل با توجه به میزان سختی آنها . و پیچیدگی محاسباتی آنها

alphabet: any nonempty finite set

Σ and Γ — برای نشان دادن الفبا

① الفبا: یک مجموعه‌ی متناهی ـ ناتهی از سمبل‌ها است.

(حرف) سمبل: اعضای تشکیل دهنده‌ی این الفبا

②

$\Sigma_1 = \{0, 1\}$ $\Sigma_2 = \{a, b, c, d\}$

$\Gamma = \{0, 1, x, y, z\}$

③ String over an alphabet: finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas.

رشته: رشته‌ی متناهی از سمبل‌ها که روی یک الفبا هستند که در کنار هم نوشته می‌شوند (الحاقی)

if $\Sigma_1 = \{0, 1\}$ then 01001 is a string over Σ_1

$\Sigma_2 = \{a, b, c, \dots, z\} \rightarrow$ abracadtby is a string over Σ_2

④ length of w = $|w|$ = number of symbols that it contains.

طول رشته = تعداد سمبل‌های که در بر دارد

⑤ The string of length zero is $\boxed{\varepsilon}$: called the empty string

رشته‌ی تهی با طول صفر — خالی — $\boxed{\lambda \text{ یا } \Delta}$ یا

empty string plays the role of $\boxed{0}$ in a number system.

⑥ if w has length n, we can write $w = w_1 w_2 \ldots w_n$
where each $w_i \in \Sigma$ w حرفها

⑦ The reverse of w, is $\boxed{w^R}$ → is the string obtained by writing w in the opposite order
(i.e., $w_n w_{n-1} \ldots w_1$) کاراکترها را برعکس بنویسیم.

⑧ string z is a substring of w if z appears consecutively within $\boxed{w}$.
abra$\boxed{\text{cad}}$abr مثلاً cad یک زیر رشته است از
پشت سر هم شده

⑨ string x → length $= m$ concatenation of x and y $= \boxed{xy}$
string y → " $= n$
با اضافه کردن y به آخر x بدست میاد.
($x_1 x_2 \ldots x_m y_1 \ldots y_n$)

⑩ to concatenate a string with itself many times → x^k $\underbrace{xx \ldots x}_{k}$
k بار x را به خودش
می‌چسبانیم.

⑪ lexicographic قانون →
ترتیب رشته‌ها مثل ترتیب dictionary الفبا.
ما از یک قانون دیگر به نام shortlex استفاده می‌کنیم.
ترتیب قانونی مطابق الفبا اما کلمات کوتاه‌تر اولویت بیشتری دارند. → The string ordering of all strings over the alphabet $\{0,1\}$ is $(\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots)$
رشته تهی   طول یک   رشته با طول ۲

پیشوند

(۱۲) x is a prefix of y if → x اول y باشد
یک z وجود داشته باشد که xz = y

suffix : پسوند → اگر رشته‌ی x آخر رشته‌ی y ظاهر شود

(۱۳) A language is a set of strings.

prefix-free if — زبان = مجموعه‌ای از رشته‌ها

no member is a propper prefix of another member.

الفبا ← متناهی — رشته ← متناهی — زبان‌ها ← لزوما متناهی نیستند

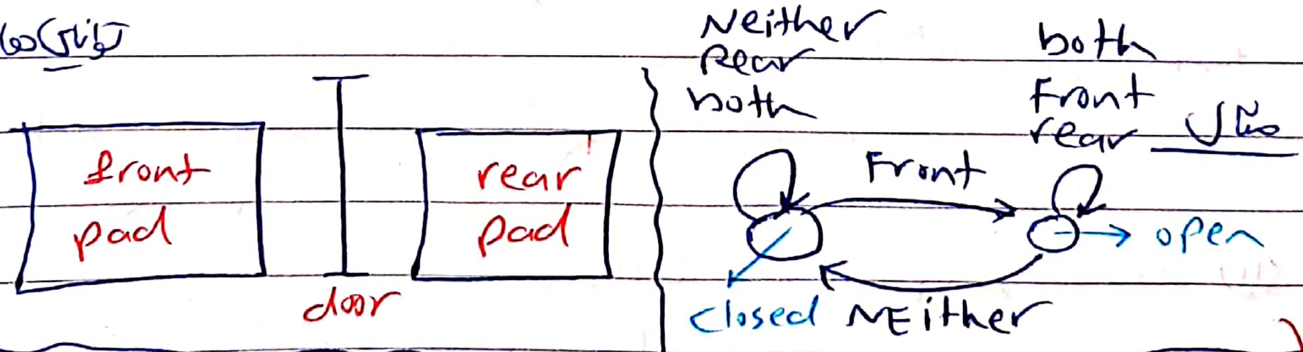
مدل‌هایی که معرفی می‌شوند abstrac هستند.

we begin with the simplest model called the finite state machine or finite automation

مدل‌های خوبی برای کامپیوترها هستند

مقدار خیلی کم و محدودی memory دارند — finite automata

چه کارهایی می‌توانند بکنند و چه کارهایی نمی‌توانند بکنند؟ محدودیت‌ها و توانایی‌هایشان؟



برای نشان دادن اینکه در [ ] توی چه حالتی (state) ای هست به یک بیت نیاز داریم

۲ تا pad داریم ← ۴ تا سیگنال می‌توانند ارسال کنند

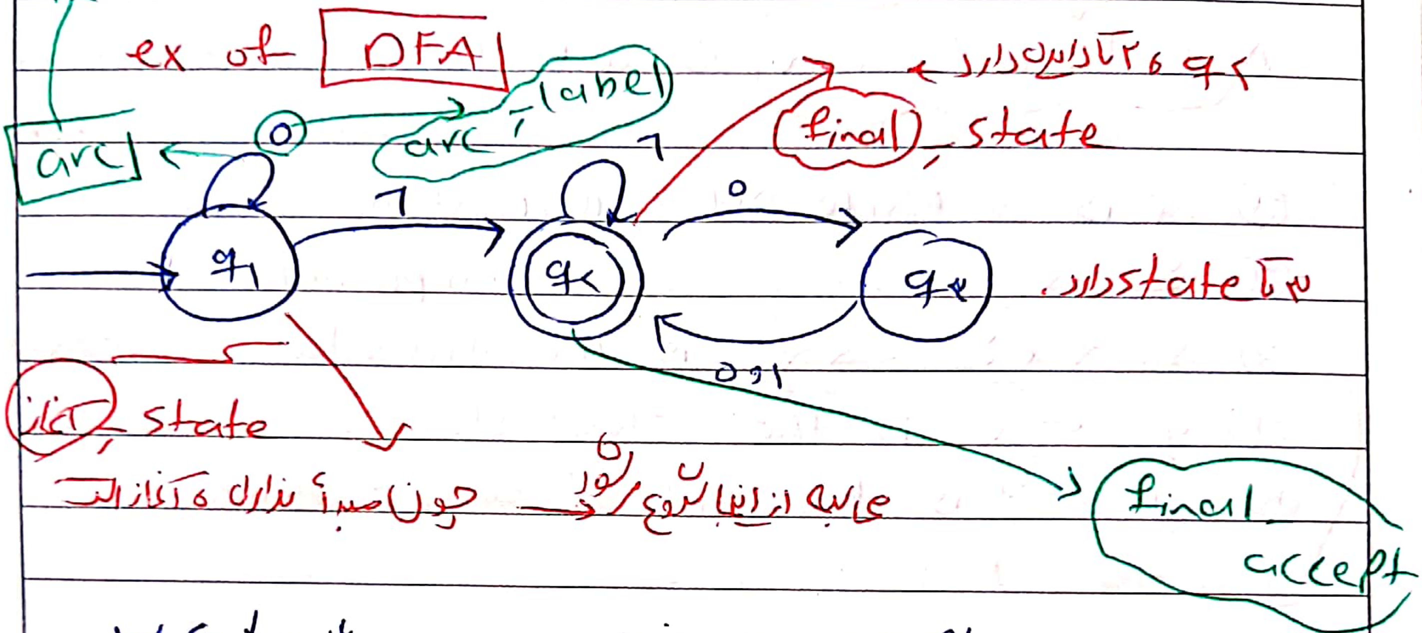
state diagram

state	Neither	front	Rear	Both
closed	closed	open	closed	closed
open	closed	open	open	open

transition

یال جهت‌دار

ex of DFA



چون مبدأ ندارد و آغاز است

محاسبه از اینجا شروع می‌شود

$w \in \{0, 1\}$

ورودی

خروجی → accept / reject

$1101 \rightarrow q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_2$

تک تک حرف‌ها

حرف ورودیم

چون کارمون به q_2 ختم شد

و چون q_2 یک state accept هست و

این رشته قبول می‌شه

کار دستگاه بالا چیه؟ تعدادی رشته را قبول می‌کنه و تعدادی را رد می‌کنه

DFA → Deterministic Finite automata

قطعی

در این ماشین‌ها رفتارها قطعی است.

برای اینکه رشته‌ای توسط DFA بالا قبول شه باید

حداقل یه یک باشه

۱) نباید تمام صفر باشه

۲) بعد از آخرین ۱

یا زوج تا صفر داریم یا صفری نداریم

زبان تقلید M_1 ← مجموعه‌ی رشته‌هایی که توسط DFA پذیرفته می‌شوند

ماشین → $\boxed{L(M_1)}$

$\boxed{\text{Session 2}}$

«formal definition of finite automata»

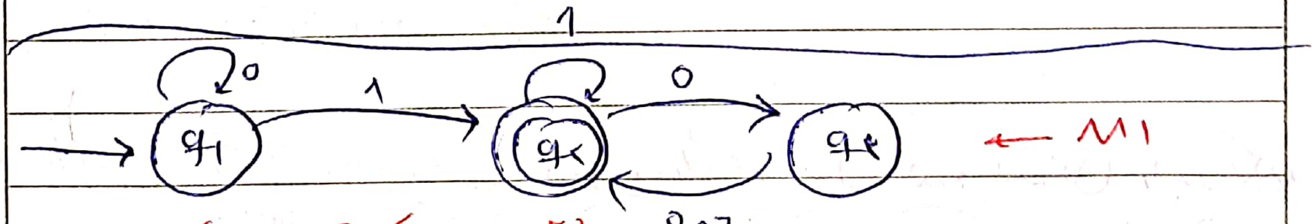
A finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
n تایی مرتب

1. Q is a finite set called states
2. Σ is ~ ~ alphabet
3. $\delta : Q \times \Sigma \to Q$ is the transition function (دامنه، برد، ضرب دکارتی)
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states (زیرمجموعه F → final)

$\delta(q_0, a) = q_3$ اگر در q_0 باشیم و a را ببینیم به q_3 میرویم

که کجا هستیم الان — کاراکتری که می‌خوانیم چیست

★ δ (تابع) باید مشخص کند به ازای هر کدوم از الفبا اگر توی یک state ، این کاراکتر را خواندیم به کجا باید بریم



← M_1

$M_1 = (Q, \Sigma, \delta, q_1, F)$

$Q = \{q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

q_1 is the start

$F = \{q_2\} \subseteq Q$

δ is described as → جدول

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

$\delta(q_1, 1) = q_2$

طبق تعریف δ ، تغییر هر عضو الفبا دقیقاً یک خروجی

نکته: در دیاگرام DFA لازم هست تغییر هرکدام از اعضای الفبا یک arc خروجی از هر state داشته باشیم ←

در q_1 ← تغییر صفر یک خروجی ... در q_2 ← تغییر صفر یک خروجی

به یک ... به یک ...

DFA ← قبول کردن تعدادی رشته‌ها و رد کردن بقیه

if A is the set of all strings that machine M accepts → we say that A is the language of machine M → $L(M) = A$ — زبان تشخیص M

M recognizes A or M accepts A

L(M) ← M ، A را می‌پذیرد / M ، A را تشخیص می‌دهد

A machine may accept several strings, but it always recognizes only one language. if the machine accepts no strings, it still recognize one language — namely the empty language $\varnothing$

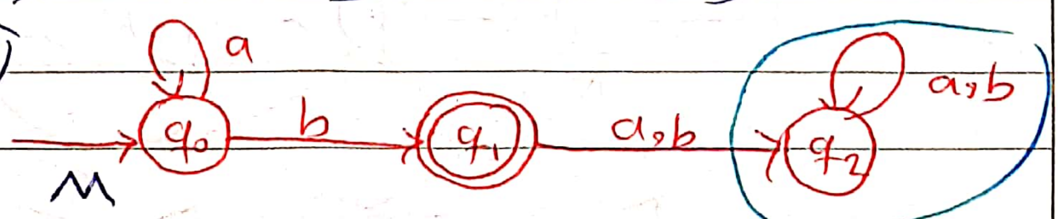
مجموعه‌ای همه رشته‌هایی که M آنها را قبول می‌کند

$A = \{w \mid w$ contains at least one 1 and an even number of 0s follow the last 1$\}$.

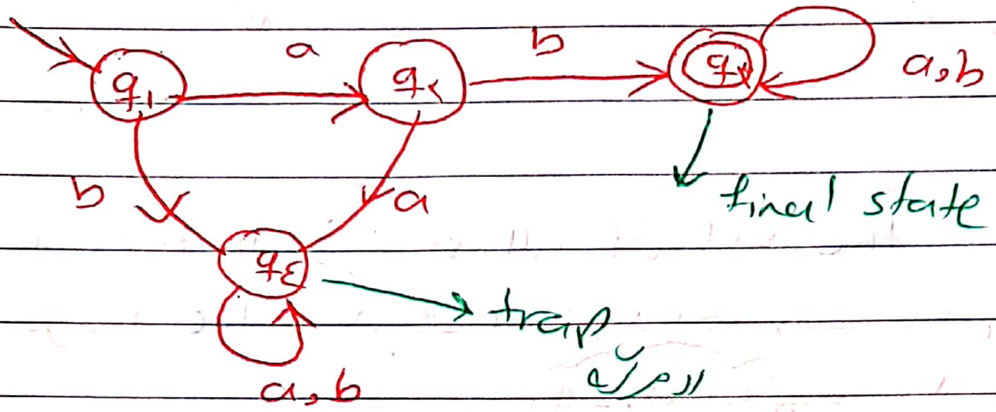
then → $L(M_1) = A$

اگر M هیچ رشته‌ای را قبول نکند ← زبان تشخیص M ، $\varnothing$ می‌شود

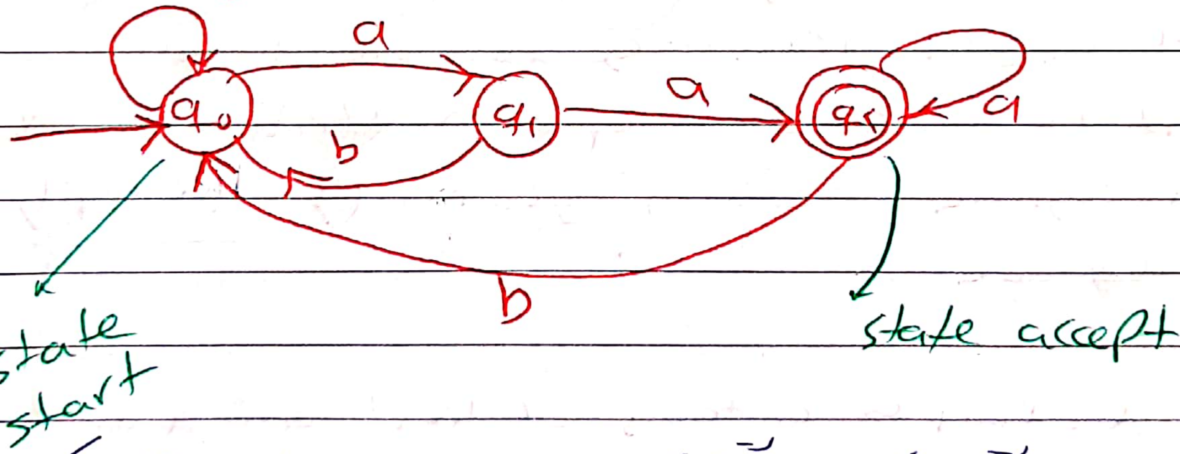
example



$\Sigma = \{a, b\}$ $L(M) = \{a^i b \mid i \in \mathbb{Z}^{\geq 0}\}$ trap ex) DFA $\Sigma=\{a,b\}$ رشته‌ای که اگر اولش ab باشد، قبول و اگر نباشد رد، یعنی ab، prefix باشد.



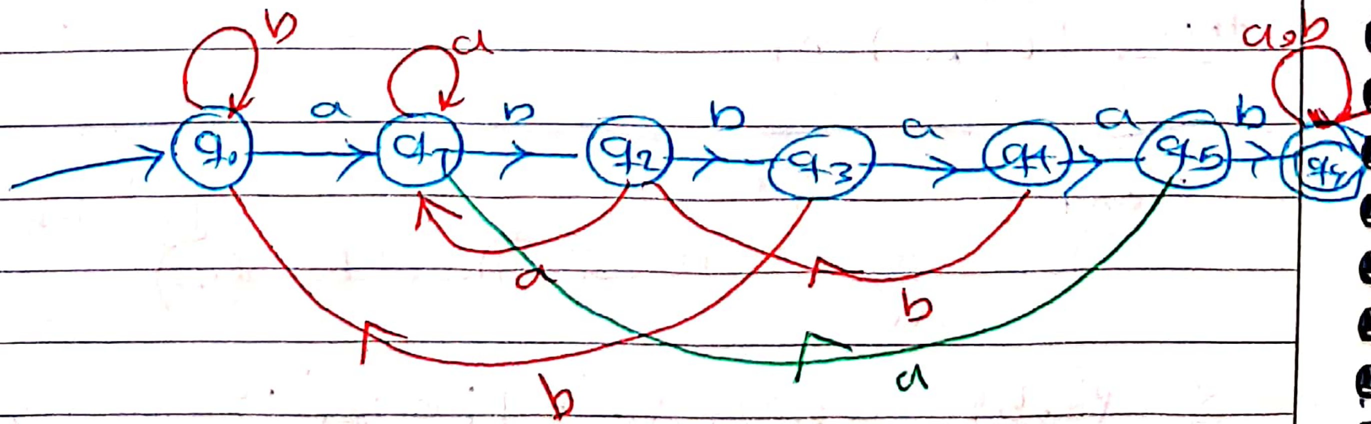
ex) رشته‌هایی که suffix آنها aa است.



همه رشته‌هایی که زیررشته‌ی abbaab را دارند قبول می‌کند

substring

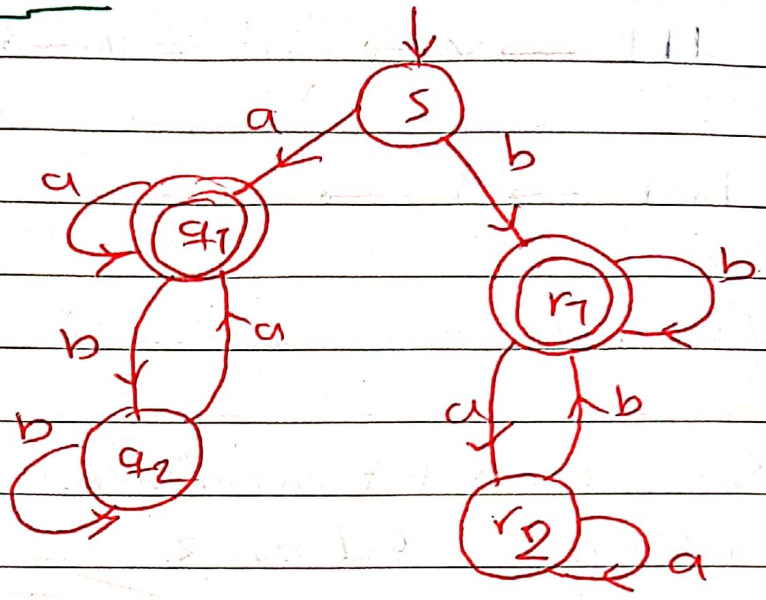
ex)



(کن reject ← یعنی trap)

ex: strings that start and end with the same symbol.

$\Sigma = \{a, b\}$

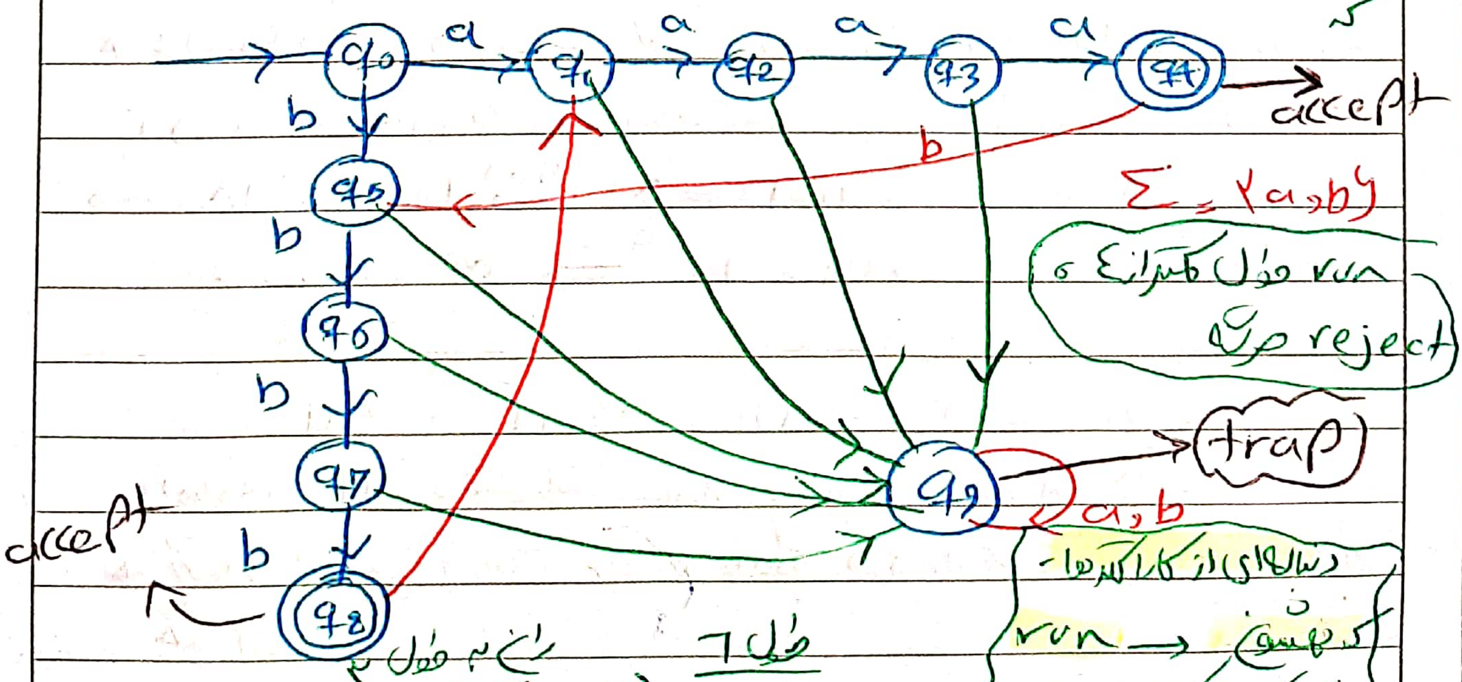


اگر اول b دیدیم و تا زمانی که b دیدیم در فاینال state هستیم و به محض دیدن a از final state خارج می‌شویم و تا وقتی که دوباره b نبینیم به r1 برنمی‌گردیم

زمانی قبول می‌شود که کاراکتر اول با آخری یکی باشد

Example (no runs of length less than four)

رشته‌های (run های به طول کمتر از ٤ ندارند) که



$\Sigma = \{a, b\}$

run طول کمتر از ٤ ← reject می‌شود

دنباله‌ای از کاراکترها

run ← که همه‌شان کاراکتر یکسان دارند

aaaa bb aaaa ba

run به طول ٢   run به طول ٢

مثال از کتاب martin

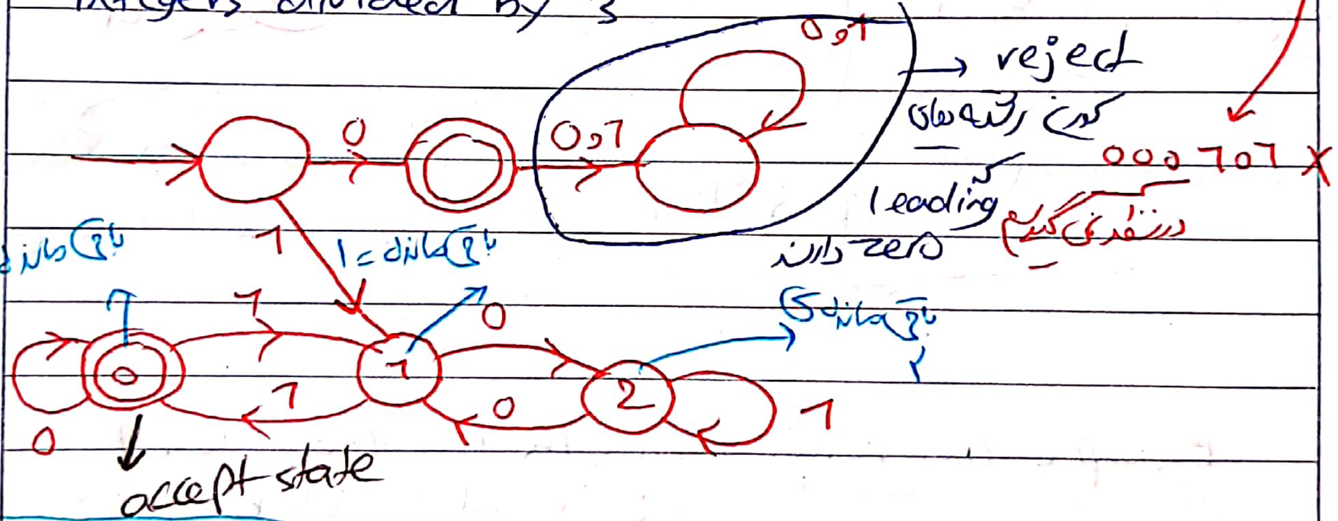
ex رشته هایی که فرم دسیمال آنها بر ۳ بخش پذیر باشد قبول در غیر این صورت رد

111 → ۷ → باقی مانده بر ۳ = ۱ → reject

1001 → ۹ → ۳ = ۰ → accept

leading zero ← صفرهایی که سمت چپ عدد هستند و بدون خواندن آنها رد می کنیم

مثال ← an FA accepting binary representations of integers divided by 3



به ازای هر صفر به سمت راست اضافه می شود عدد ۲ برابر می شود

* با اضافه کردن یک صفر در سمت راست یک عدد باینری آن عدد در مبنای دسیمال ۲ برابر می شود

1001 = ۹ $\xrightarrow{9\times 2}$ 10010 = ۱۸ →

100100 $\xrightarrow{9\times 4}$ ۳۶ و ...

$a \overset{3}{\equiv} b \rightarrow 2a \overset{3}{\equiv} 2b$, $2a+1 \overset{3}{\equiv} 2b+1$

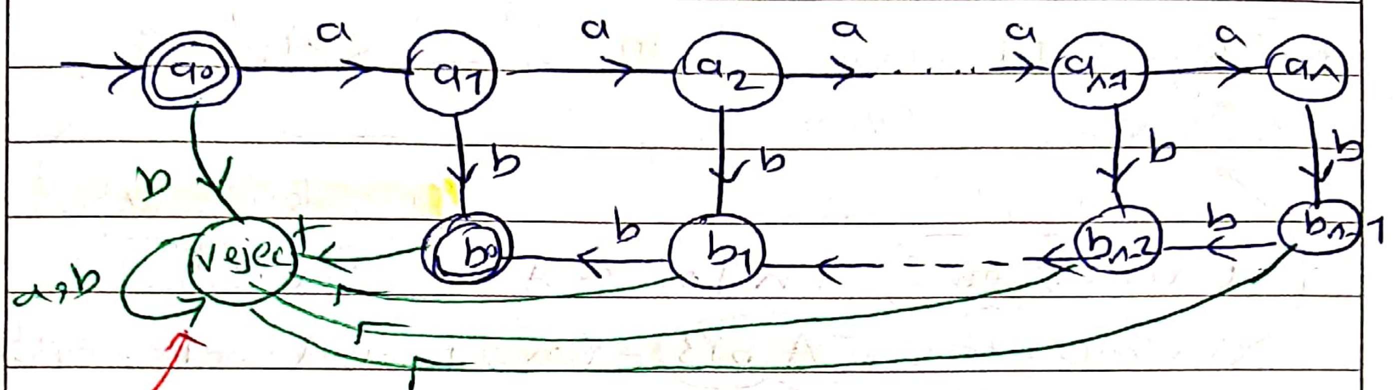
با اضافه کردن $\boxed{1}$ به سمت راست ← عدد ۲ برابر می شود و به علاوه ی یک

* رشته را از چپ به راست می خوانیم و طبق هر کدام از کاراکترها تصمیم می گیریم چه کار کنیم.

* اگر $\{a^i b^i\}$ ← این را با DFA نمی‌توان حل کرد باید با علم یا مکینت حل کرد *

ex $\{a^i b^i \mid i \le n\}$ قبول کردن همه‌ی رشته‌های به فرم

$\Sigma = \{a, b\}$ $\boxed{n}$ یک ثابت داده شده است



the incomplete specified DFA

states a_K count the number of $\boxed{a's}$

~ b_K ~ ~ $\boxed{b's}$

DFA

تعریف رسمی محاسبه برای DFA ها

Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of the alphabet Σ.

کاراکتر اول — کاراکتر آخر

Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three condition

1) $r_0 = q_0$

2) $r_n \in F$

3) $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, \ldots, n-1$

شرط‌ها

مجموعه‌ی state ها

روندها باید از q_0 آغاز شود — state

state start

$\delta(r_0, w_1) = r_1$

اگر در r_0 باشیم و w_1 را بخوانیم میریم توی r_1

regular

We say that M recognizes language زبان منظم A if $A = \{w \mid M \text{ accepts } w\}$.

A language is called a regular if some finite automaton recognizes it ← شرط منظم بودن یک زبان

زبان منظم است ← اگر نظیر یک زبان یک DFA داشته باشیم

هر رشته از A را به هر رشته از B می‌چسبانیم

عملگرهای regular

اجتماع

Union : $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Concatenation : $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

Star : $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

unary عملگر

تعداد اعضای | تعداد اعضای

تعداد اعضای مجموعه $|A \circ B| = |A| \times |B|$

* the empty string $\boxed{\varepsilon}$ is always a member of A^*, no matter what A is.

کار A^* ← کارش چسباندن رشته‌های داخل زبان A به اندازه‌ی k بار به یکدیگر

صفر بار چسباندن ← رشته تهی ← همواره عضوی از A^*

صفر بار good

را به هم bad

زبان ← مجموعه‌ای از string ها

ex alphabet (Σ) → be the standard 26 letters $\{a, b, \ldots, z\}$

if $A = \{good, bad\}$ and $B = \{boy, girl\}$,

$A \cup B = \{good, bad, boy, girl\}$

$A \circ B = \{goodboy, goodgirl, badboy, badgirl\}$

$A^* = \{\varepsilon, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, \ldots\}$

یکبار کنار هم ... دوبار کنار هم

session 3

مجموعه‌ی زبان‌های منظم، بسته است.

Theorem: The class of regular language is closed under the union operation. In other words if A_1, A_2 are regular languages, so is $A_1 \cup A_2$

① یعنی اجتماع بسته است ، یعنی زبان منظم تحت عملگر اجتماع بسته است.
یعنی اگر A_1 و A_2 زبان های منظم باشند ← $A_1 \cup A_2$ هم زبان منظم است.

$A_1, A_2 \rightarrow$ regular languages → یک DFA دارند

M_1 recognizes $A_1 \rightarrow L(M_1) = A_1$
M_2 " $A_2 \rightarrow L(M_2) = A_2$

حالا می‌خواهیم اثبات کنیم اجتماع هم منظم است که یک DFA طراحی کنیم یا نشان بدهیم DFA ای وجود دارد که زبان تشخیص آن، $A_1 \cup A_2$ است.

$L(M) = A_1 \cup A_2$

* در DFA ها فقط می‌توانی رشته را یک بار از چپ به راست بخوانی * و بعد از خواندن آخرین کاراکتر بفهمیم که نتیجه می‌گیریم که آیا رشته پذیرفته می‌شود یا نه!!

پس باید یک کاری کنیم که M همزمان A_1 و A_2 را همزمان بخواند

if M_1 has K_1 states and M_2 has K_2 states
→ the number of states one from M_1 and the other from M_2 → $K_1 \cdot K_2$ ← در بدترین حالت

(O , O)
متعلق به A_1 ، متعلق به A_2

* به شکل همزمان simulate می‌کنیم ، M_1 و M_2 را

$\Sigma_1 \cup \Sigma_2$

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

$\Sigma_1 \cup \Sigma_2$ — زوج مرتب q_1, q_2

(q_1, q_2)

$M = (Q, \Sigma, \delta, q_0, F)$

① $Q_1 \times Q_2$ — ضرب دکارتی دو مجموعه

مجموعه‌ای که مولفه‌ی اول از Q_1 و مولفه‌ی دوم از Q_2

مجموعه‌ی همه زوج مرتب‌هایی که عضو اول از Q_1 الف، عضو دوم از Q_2 الف.

$L(M) = A_1 \cup A_2$

این state هایی را قبول می‌کنیم که یا M_1 قبول کرده یا M_2

① یا مولفه‌ی اول توی F_1 هست ، یا هردو

② یا مولفه‌ی دوم در F_2 هست

③ $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

یا مولفه‌ی اول توی F_1 باشد — یا مولفه‌ی دوم توی F_2 باشد

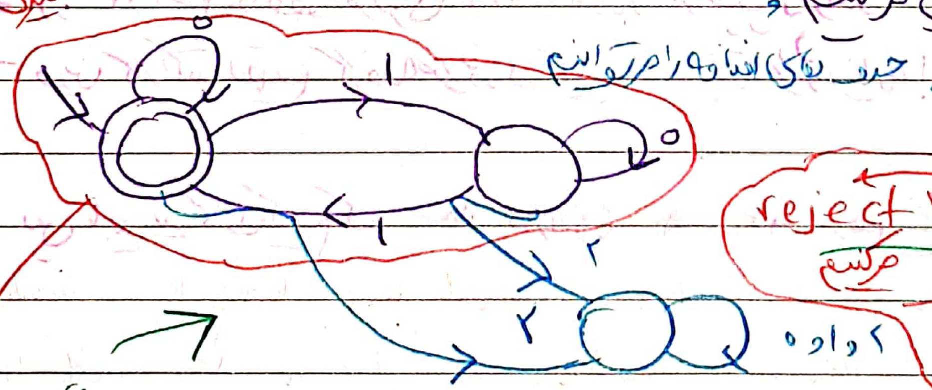
③ الفبای تغییر $\rightarrow \boxed{\Sigma_1 \cup \Sigma_2}$

اگر الفبای M_1 و M_2 تفاوت داشت چه کار می‌کنیم؟

الفباها یشان را یکی می‌کنیم و

مثل مثال زیر حرف‌های اضافه را می‌توانیم trap کنیم

الفبای تغییر: ۰ ، ۱ ، ۲ پذیرد



reject می‌کنیم — ۲ را لازم کنیم

$\Sigma = \{0, 1, 2\}$

حقیقتی الفبای M_1 را بزرگ کنیم در حالی که زبان تغییر M_1 همانی قبلی باشد

اگر الفبا واحد نبود کارهای بالا را انجام می‌دهیم، رشته‌های Q_1 و Q_2 زوج تا را دارند

Q — تابع transition

Q_1 توی Q_2 توی

④ $\delta: (Q_1 \times Q_2) \times \Sigma \rightarrow Q_1 \times Q_2$

$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

عضو Q الف — $\in \Sigma$

کجا هستیم، چه سمبلی را از ورودی می‌خوانی و کجا می‌خواهی بری

$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

وقتی که در r_1 هستی و a مرفوی ببین چه اتفاقی می‌افتد

در M_2 چه اتفاقی می‌افتد که تنها در r_2 هستی و a را مرفوی

(c) $q_0 \rightarrow (q_1, q_2)$

(A) برای اشتراک

$F = F_1 \times F_2$

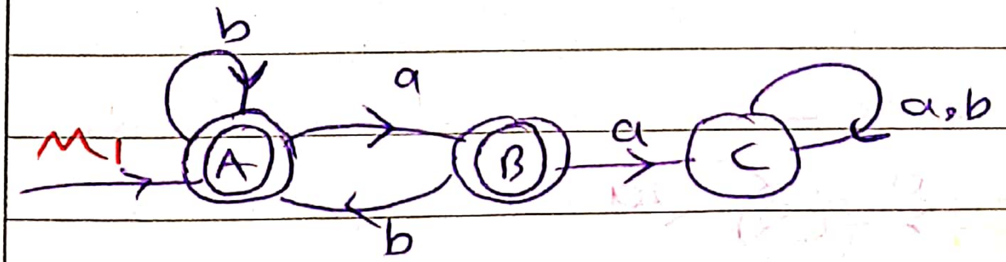
موفق‌های اول در M_1 و M_2 ~ دوم ~

رشته‌هایی که هم M_1 آنها را بپذیرد هم M_2 → $L(M) = A_1 \cap A_2$

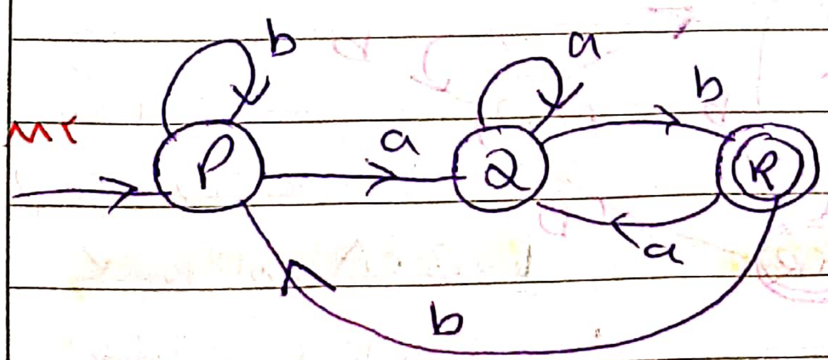
ex- سوال ۵ رشته‌های با شرایط را هم و هم accept کنیم که یا اشتراک L_1 را داشته باشند یا اشتراک L_2

$L_1 = \{x \in \{a,b\}^* \mid aa \text{ is not a substring of } x\}$ &

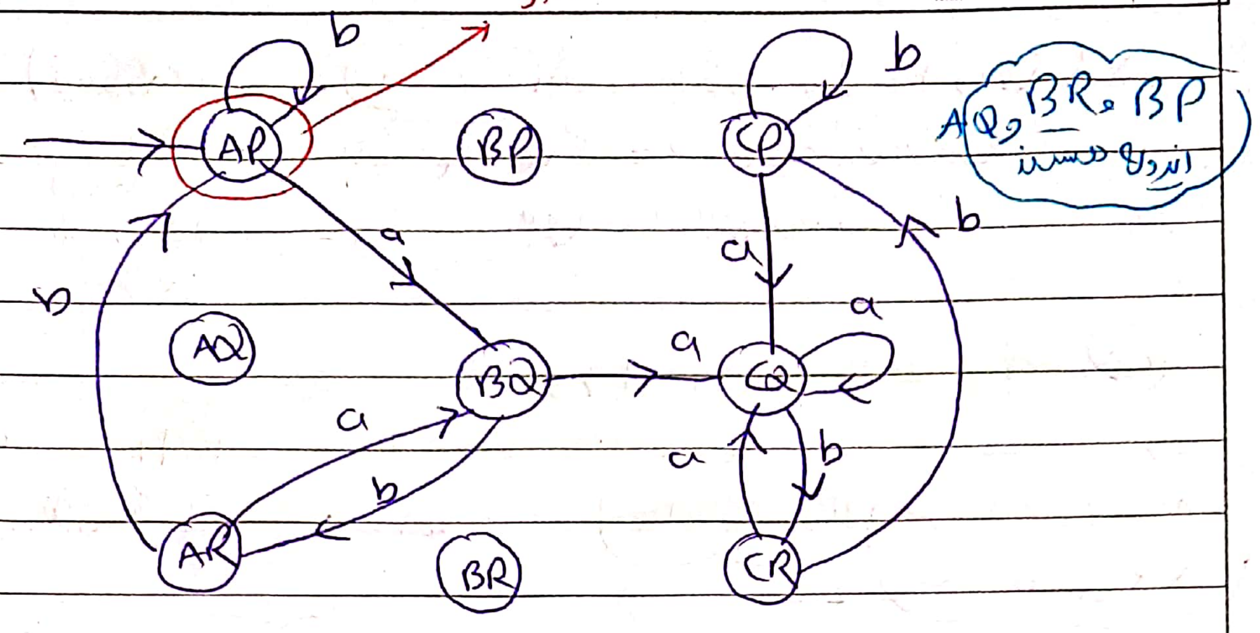
$L_2 = \{x \in \{a,b\}^* \mid x \text{ ends with } ab\}$



$M_1 \cup M_2 = ?$



start



$M_1 \cup M_2$ ؟ زوج مرتب‌هایی که مولفه‌ی اول اعضای A یا B و مولفه‌ی دوم اعضای P یا Q یا R

$3 \times 3 = 9$

در M_1 وقتی در C هستیم و a مصرف شد

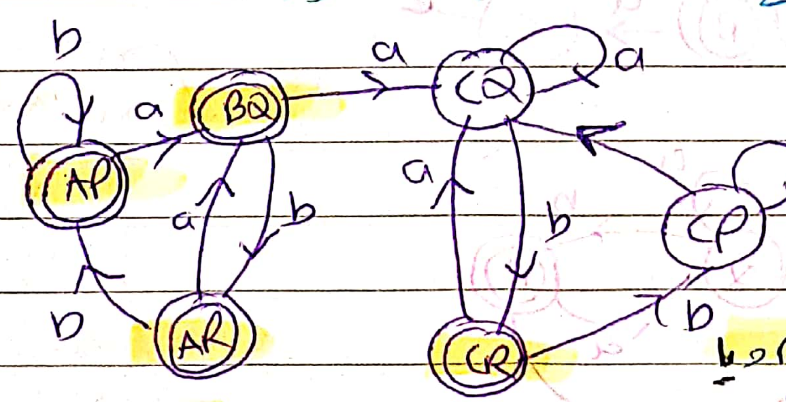
state ها در $\delta(CP, a) = (\delta_1(C, a), \delta_2(P, a))$

$= CQ$

در M_2 وقتی در P هستیم و a مصرف شد

$\delta(AR, a) = (\delta_1(A, a), a), \delta_2(R, a)) = BQ$

۳ تا state غیرقابل دسترسند ← ۶ تا می‌مونه



کدام state ها accept هستند؟

مولفه اول A یا B باشد

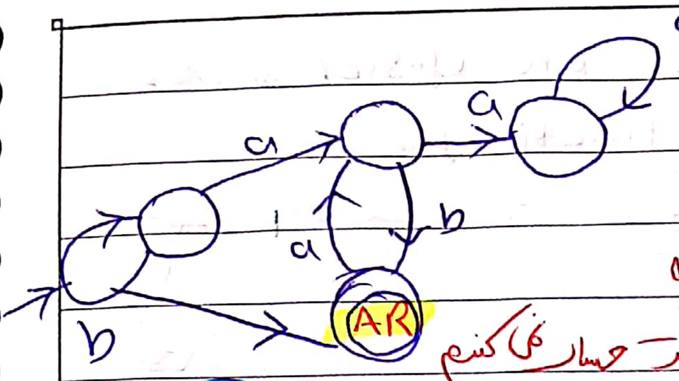
مولفه‌ی دوم R باشد

AP , BQ , AR , CR

$L_1 \cup L_2$

DFA ـ تغییر احتیاج

DFA ـ تقلید اشتراک



برای اشتراک $F_1 \times F_2$

فقط AR accept می‌شه

BR هم شرط رو داره ولی چون انبوله داره حساب نمی‌کنیم

$L_1 \cap L_2$

$L_1 \setminus L_2$ ← تفاضل دو مجموعه ← $L_1 - L_2$

عملگر باینری:

$L_1 \cup L_2$ $\quad$ $L_1 \cap L_2$ $\quad$ $L_1 \setminus L_2$ (تفاضل) $\quad$ $L_1 \circ L_2$ (concat)

دو عملگر unary:

$\overline{L_1}$ ← مکمل ← complement $\quad$ L_1^*

complement

مکمل یک زبان:

the complement of $L \subseteq \Sigma^*$ denoted by $\overline{L}$, is such that $\overline{L} = \Sigma^* \setminus L$

همه رشته‌هایی که روی الفبا قرار دارند ولی داخل زبان ان الفبا قرار ندارند

همه‌ی رشته‌هایی که داخل L قرار ندارند

مثال: اگر L دارای همه رشته‌هایی باشد که طول آنها زوج است

$\overline{L}$ ~ ~ ~ است ~ فرد ~

Σ^* → مجموعه همه رشته‌هایی که روی الفبای Σ تعریف می‌شوند

اگر L منتظم باشد ← $\overline{L}$ هم منتظم است Final ها را non final کن و برعکس

$M = (Q, \Sigma, \delta, q_0, F)$ $\quad$ $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$

→ $L(M') = \Sigma^* \setminus L = \overline{L}$

$Q - F$ را به‌عنوان final ها قبول می‌کنیم ← اگر زبان تقلید M، L باشد $(L(M) = L)$

the regular languages are closed under set difference (subtraction)

$L_1 - L_2 = L_1 \cap \overline{L_2}$ → با اشتراک بسته

منظم الف

منظم   منظم

راه کلی برای اثبات منظم بودن اشتراک :

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

$L_1 \to \overline{L_1}$
$L_2 \to \overline{L_2}$ $\Big\} \to \overline{L_1} \cup \overline{L_2} \to \overline{\overline{L_1} \cup \overline{L_2}}$ منظم →

منظم

$L_1 \cap L_2$ منظم است

اثبات بسته بودن یک زبان تحت دو عملگر * و o هم می‌توان گفت :

با non deterministic اثبات میکنه

Nonedeterminism & NFAs

session 4

① DFA ها یا Deterministic computations

when the machine is in a given state and reads the next input symbol, we know what the next state will be → it is determined

رفتار قطعی / می‌دانیم در هر state که هستیم بعد کجا می‌خواهیم برویم

غیرقطعی

② in a nondeterministic machine, several choices may exist for the next state at any point.

که ممکن است تعریف شده باشد چند خروجی داشته باشیم یا در state‌های هیچ خروجی نداشته باشیم

ممکن است چند تا اتفاق بیفتد

③ Nonedeterminism is a generalization of determinism, so every DFA is automatically an NFA

① NFA ها قدرت بیان خیلی بیشتری دارند نسبت به DFA ها → چرا از NFA ها موقع استفاده کنیم وقتی غیرواقعی هستند !؟ ②

② طراحی NFA یک زبان خیلی راحت تر است نسبت به DFA

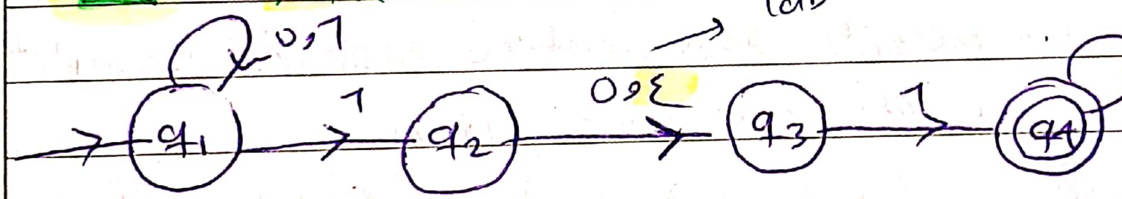
regular

③ تعداد state ها خیلی کمتر است و راحت تر درک می شود

④ از آنجایی که هر NFA می تواند به DFA تبدیل شود، با ساختن یک NFA می توانیم خیلی راحت یک DFA را بسازیم.

در DFA تحت هر سمبل و از هر state ای دقیقا یک خروجی داریم ✗

Ex) NFA



① در NFA ، تحت یک state ، می توانیم یک خروجی داشته باشیم یا بیش از
q2 یا هیچ خروجی نداشته باشیم

لیبل = ε

② یک arc داریم که ε است

تحت هر سمبل هیچ تا خروجی داشته باشیم

منطقی عبارات را انجام می دهد

How does an NFA compute?

① we are running an NFA on an input and come to a state with multiple ways to proceed. after reading that symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel.

each copy of the machine takes one of the possible ways to proceed and continues as before. if there are subsequent choices, the machine splits again.

تقسیم هر خروجی (حتی صفر) اگر ۰ تا خروجی داشته باشیم، باید ببینیم چه اتفاقی می‌افتد.

③ if the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy of the machine dies, along with the branch of the computation associated with it.

تقسیم یک state ای که به ازای یک سمبل خروجی نداشته باشیم

④ if any one of these copies of the machine is in an accept state at the end of the input the NFA accepts the input string.

اگر یکی از این کپی ها / اصطلاحاً هایی که داریم ما را به یک accept state ببرد ما رشته ورودی را قبول می‌کنیم در غیر این صورت نه. (چون زمانی یک رشته‌ی ورودی را قبول یا رد می‌کنیم)

★ اگر روی label یک arc، ε است به معنی اینه به ازای صرفاً arc داریم می‌توانیم در انتهای arc هم باشیم.

if a state with an ε symbol on an exiting arrow is encountered, then without reading any input, the machine splits into multiple copies

↑ وقتی با یک $\boxed{\varepsilon}$ مواجه شویم.

NFA درخت محاسبات

Tree of possibilities   درخت احتمال‌ها

Computation Tree   درخت محاسبات

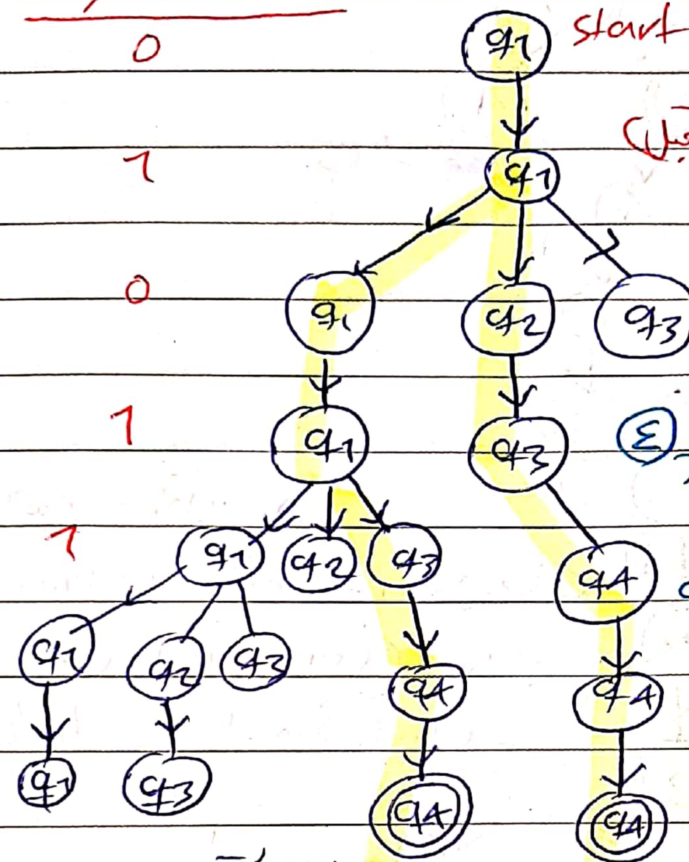
Symbol read

0
1
0
1
1
0

start

N1 → 010110

رشته ورودی مثال قبلی (دو صفحه قبل)



چند تا امکان داریم

وقتی در q_2 هستیم چون ε arc داریم

می‌تونیم بدون خوندن سمبل به q_3 بریم

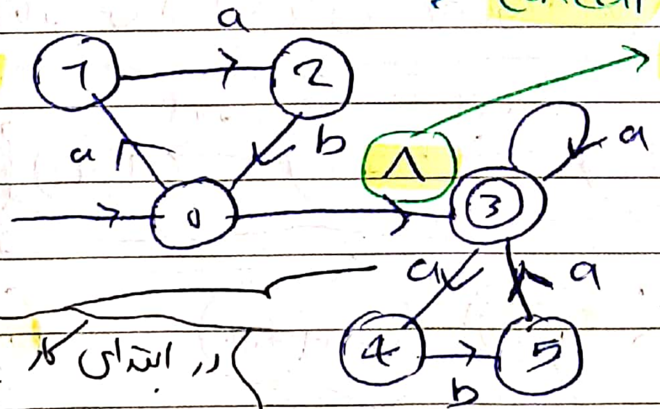
نماد تهی ε

همه رشته‌هایی که با 101 یا 11 را به عنوان زیر رشته دارند.

Ex $(aab)^*(a, aba)^*$ concat

یا می‌خوانیم a یا aba

رشته تهی



input string e

aababa

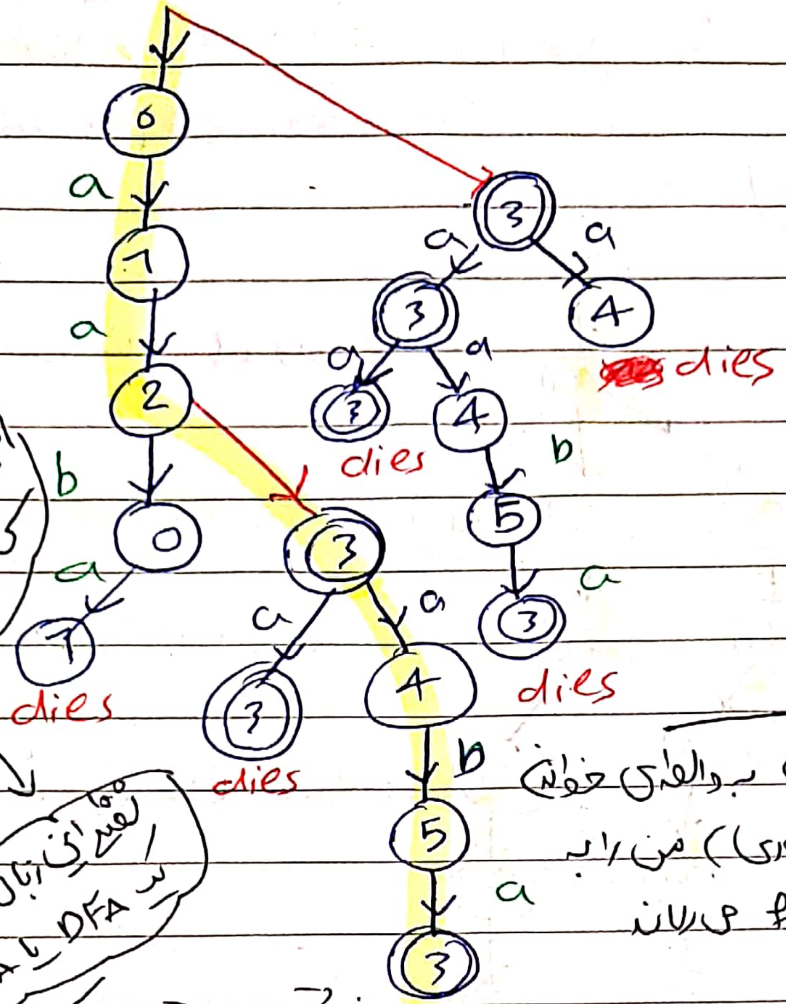
در ابتدای کار یا توی صفر هستیم یا ۳

اگه توی 2 باشیم، b می‌خوانیم

یا

میریم به صفر

میریم به 3



همه ی الفبای تهش که سمبل سوم از آخرش ۱ است

تفسیری زبان ها که NFA یا DFA هم ارزند

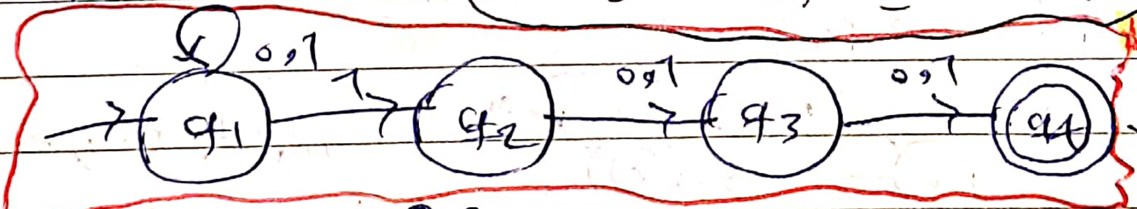
چون حداقل یک branch هست که من بعد الفای خوندن

تن (درخت کی دوری) من را به

final state می رسانه

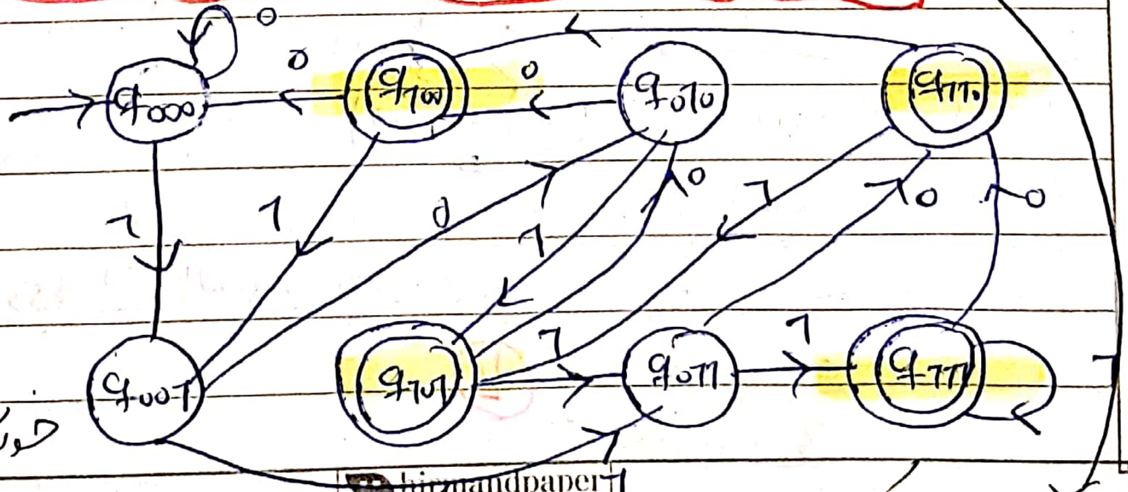
NFA ـ L۲ و این الگو را قبول می کند

ex) A is language consisting of all strings over {0,1} containing a 1 in the third position from the end. اثبات کن که این زبان regular است



DFA ـ

زبان ـ DFA



خورش را می رسانه به q_4

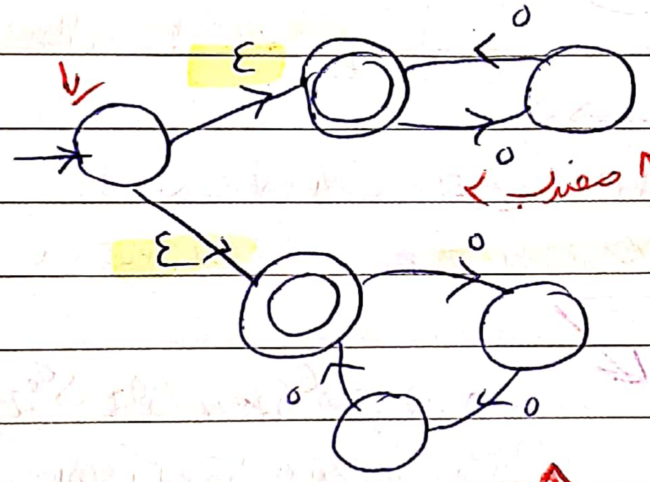
هر مقداری می خونه و وقتی به سومی بید از آخر رلیی ← از q_2 رد می شود

درهایی که اگر سمبل سوم از آخره ۱ نباشه ← ۰ شوه از ۹۶ را شود

every NFA can be converted into an equivalent DFA

the smallest DFA for A contains 8 states.

ex) this machine demonstrates the convenience of having ε arrows. it accepts all strings of the form of $\boxed{0^k}$ where k is a multiple of $\boxed{2 \text{ or } 3}$ it accepts strings $\varepsilon, 00, 0000,$ ___ but not $0, 00000$



مضرب ۲ ↑

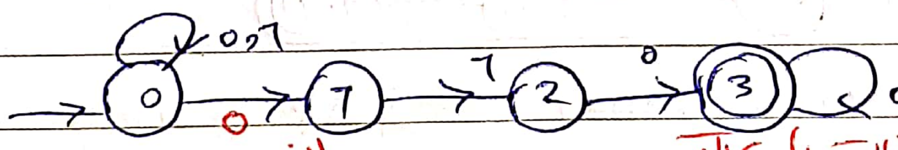
یا مضرب ۲ باشه
یا مضرب ۳ باشه

یا هردو

مضرب ۳ ↑

پیدا کردن رشته زیررشته

ex) find an NFA that accepts the set of binary strings having a substring 010.



سمبل که می‌خوانیم

یا کمتر از الفبا یا ε الف

الفبای

مجموعه‌ای state‌ها

"تعریف رسمی یک NFA"

a nondeterministic finite automaton is a 5_tuple

$(Q, \Sigma, \delta, q_0, F)$ → موقعیت آغاز

① Q is a finite set of states ② Σ is a finite alphabet

transition تابع final state

③ $\delta : Q \times \Sigma_\varepsilon \longrightarrow P(Q)$ is the transition function

⑤ $q_0 \in Q$ is the start state

⑥ $F \subseteq Q$ is the set of accept states

▲ for any set Q we write $P(Q)$ to be the collection of all subsets of Q.
it can also be denoted by $\boxed{2^Q}$ → here $P(Q)$ is called the power set of Q.

$\Sigma_\varepsilon \longrightarrow \Sigma \cup \{\varepsilon\}$

تابع انتقال یک DFA

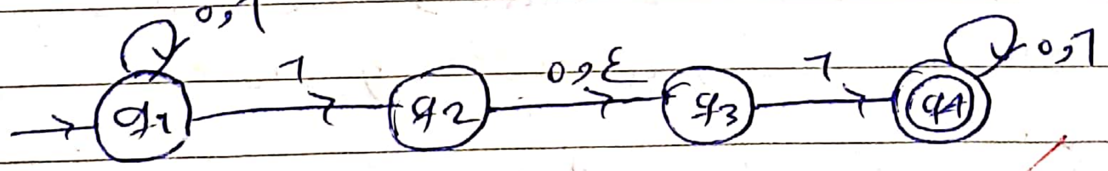
$\delta : Q \times \Sigma \longrightarrow Q$

در یک NFA ممکن است با خواندن یک سمبل چندجا بریم یا هیچ جا نریم
چنین امکان دارد ← یک زیرمجموعه از Q است نه یک عضو

powerset → مجموعه همه زیرمجموعه‌های Q

$P(Q)$ یا 2^Q → در تابع transition

← جاهایی که می‌توانیم بریم یک زیرمجموعه از Q است نه عضوی از Q



به عنوان مثال الف

① $Q = \{q_1, q_2, q_3, q_4\}$

② $\Sigma = \{0, 1\}$ ③ δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

④ q_1 is the start state

⑤ $F = \{q_4\}$

هم‌ارزی DFA ها و NFA ها

Deterministic and nondeterministic finite automata recognize the same class of languages.

① two machines are equivalent if they recognize the same language

② an NFA that accepts some language L → there exists an equivalent DFA that accepts L

M_1 و M_2 هم‌ارز هستند اگر $\boxed{L(M_1) = L(M_2)}$

③ هر DFA یک NFA هم هست

هر زبانی که توسط DFA قبول شود توسط NFA هم قبول می‌شود

روش دوم / بستار ε

the ε-closure of a set of states

★ اگر زبانی توسط NFA پذیرفته شود، یک DFA هم دارد ★

$M = (Q, \Sigma, \delta, q_0, F)$ is an NFA and $R \subseteq Q$ is a set of states,

the ε-closure of R is the $E(R)$ that can be defined recursively as follows (بستار $=E(R)$)

① $R \subseteq E(R)$ stateهایی که از R با مسیر ε بتوان به آن‌ها رسید

② for every $q \in E(R)$, $\delta(q, \varepsilon) \subseteq E(R)$ — ε arc

زیرمجموعه‌ای از state ها

for any subset R of Q, we define E(R) to be the collection of states that can be reached from members of R by going only along ε arrows. for $\boxed{R \subseteq Q}$ → ε arrows.

$E(R) = \{q \mid q$ can be reached from R by traveling 0 or more session5

تابع انتقال تعمیم یافته

The Extended Transition Function δ^* for DFA

ها DFA ن ←

آغاز state

$\delta^* : Q \times \Sigma^* \to Q$

یک string باید

state پایانی

به جای اینکه symbol باشد

اگر $M = (Q, \Sigma, \delta, q_0, F)$ یک DFA باشد

extended function

$\delta^* : Q \times \Sigma^* \to Q$,

base ① for every $q \in Q$, $\delta^*(q, \varepsilon) = q$

② ... , every $y \in \Sigma^*$

and every $\sigma \in \Sigma$, $\delta^*(q, y\sigma) = \delta(\boxed{\delta^*(q, y)}, \sigma)$

توی هر stateای که باشی اگر یک ε بخوانی توی همان stateای که بودی می‌مانی

مثل یک تابع بازگشتی

σ = سمبل واحد

مفهوم پذیرش توسط یک DFA

اگر $M = (Q, \Sigma, \delta, q_0, F)$ یک DFA باشد و

$x \in \Sigma^*$ ، رشته x توسط M قبول می‌شود اگر $\delta^*(q_0, x) \in F$

و را مرحله در نهایت صورت

از q_0 شروع کنیم و رشته x را بخوانیم

اگر final باشد ← کار ما به state ختم می‌شود

the language accepted by M is the set

$L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$

زبان تفسیر M ، stringهایی که توسط M قبول می‌شوند

آن string را قبول می‌کنیم

if L is a language over Σ, L is accepted by M if and only if $L = L(M)$

formal define of compution for NFA تعریف رسمی مفهوم پذیرش برای

$N = (Q, \Sigma, \delta, q_0, F)$ is an NFA and w a string over the alphabet Σ. we say that N accepts w if $w = y_1 y_2 \ldots y_m$ (y_i is a member of Σ_ε) and a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with 3 conditions:

① $r_0 = q_0$ ← که یعنی اولین state باید آغاز باشد

یا عضوی از الفبای یا ε

چرا تفضویت داریم ولی در DFA نداریم → چون اینجا یک set یا مجموعه داریم

(۲) $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, 1, \ldots, m-1$

state ← یعنی عضوی از set

(۳) $r_m \in F$ — شاید: وقتی توی r_0 هستی و y_1 را میخونی

میری توی r_1 ← y_1 باید داخل r_1 باشد

به ازای خواندن یک سمبل ممکن است از مجموعه‌ای از state ها قرار بگیریم

اگر $N = (Q, \Sigma, \delta, q_0, F)$ یک NFA باشد

تابع انتقال تعمیم یافته — we define extended transition function

$\delta^* : Q \times \Sigma^* \to 2^Q$ as follows:

کجا هستی — یک string — کجاها میتونی بری

تابع انتقال تعمیم یافته ← شاید رشته را که بخونی کجاها میتونی بری

اگر هیچی نخونی ← جاهای مرسوم

(1) for every $q \in Q$, $\delta^*(q, \varepsilon) = E(\{q\})$. تعریف به شکل بازگشتی

(2) for every $q \in Q$, every $y \in \Sigma^*$, and every $\sigma \in \Sigma$

y = همه‌ی سمبل‌ها به جز آخری ← سمبل آخر + بقیه رشته

$$\delta^*(q, y\sigma) = E(\cup \{\delta(p, \sigma) \mid p \in \delta^*(q, y)\})$$

اجتماع ← ε-closure

مجموعه جاهایی که با ε خوندن میتونیم بریم

ε-closure ← q: یعنی جاهایی که با کمک ε arrow ها میتونیم بریم

A string $x \in \Sigma^*$ is accepted by N if $\delta^*(q_0, x) \cap F \neq \emptyset$

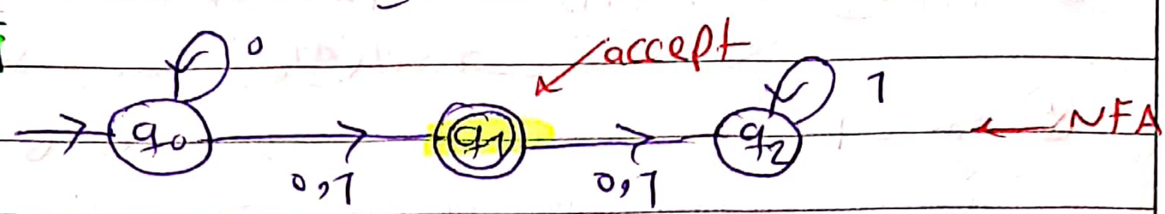
language $L(N)$ accepted by N is the set of all strings accepted by N.

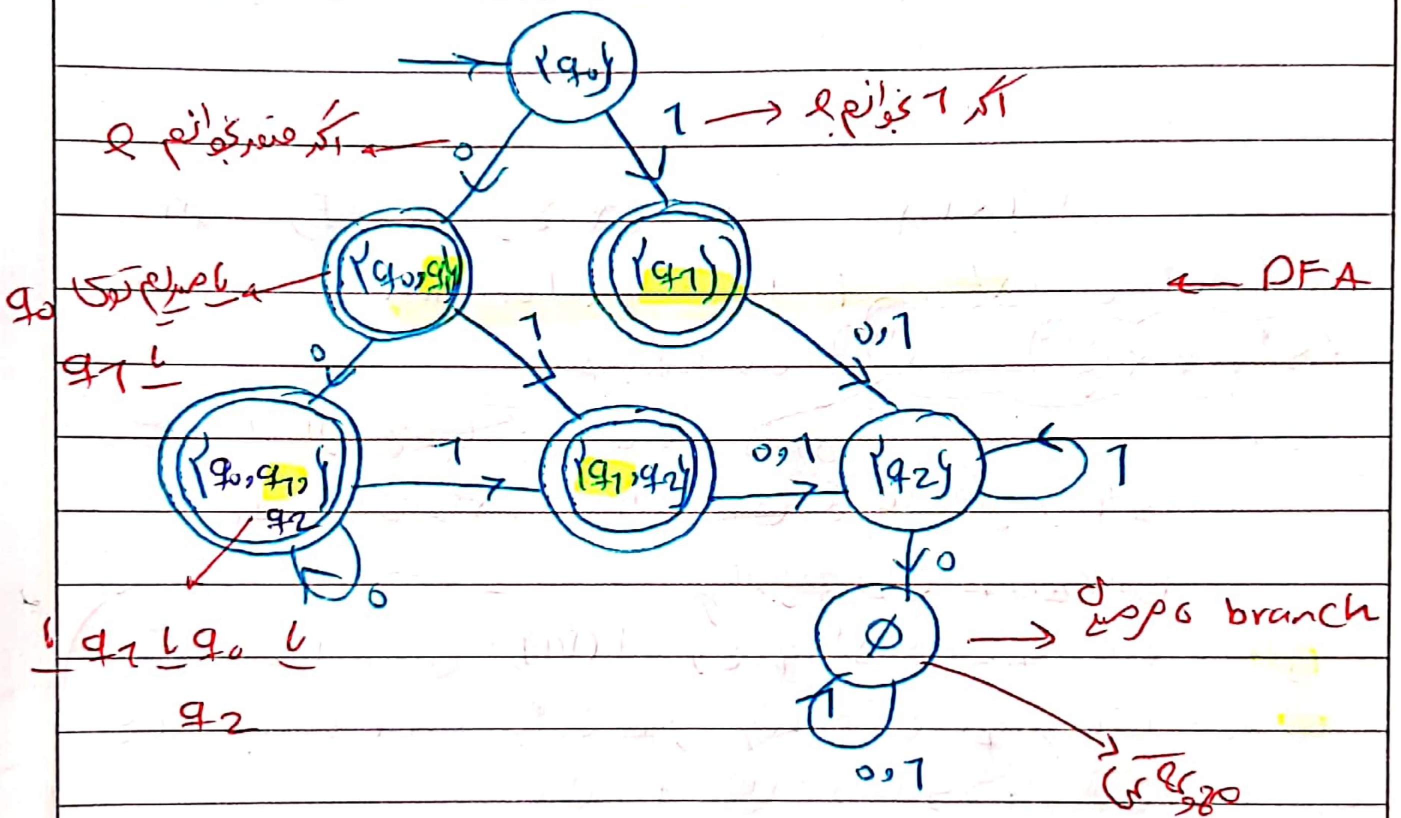
در حالت final قرار بگیریم ← x پذیرفته میشود

زبان نظیر N — مجموعه همه‌ی رشته‌هایی که توسط N پذیرفته میشود

قدرت محاسباتی NFA و DFA برابرند

$\boxed{ex}$





کدام مجموعه‌ها accept می‌شن؟ اونایی که q_1 دارند

equivalence of DFAs & NFAs

* every NFA has an equivalent DFA *

for every language $A \subseteq \Sigma^*$ accepted by an NFA $N=(Q,\Sigma,\delta,q_0,F)$, there is a DFA $M=(Q',\Sigma,\delta',q_0',F')$ that also accepts A.

اول A برای حالتی که ε arrow نداریم و اثبات می‌کنیم بعد آن $\boxed{\varepsilon}$ را هم اثبات می‌کنیم

مجموعه ای همه ی زیر مجموعه های Q

① $Q' = P(Q)$ every state of M is a set of states of N.

در بردارنده ی همه ی زیرمجموعه های مجموعه ی Q است

P(Q) is the set of subsets of Q *

پس مثلاً اگر NFA ما ۳ تا state داشت ،
در DFA ما ۲^۳ تا ، داریم

اما همگی لزوماً استفاده نمیشوند (مثلاً قابل دسترسی نباشند برای state)

② for $R \in Q'$ and $a \in \Sigma$

یکی از state های
Q'

$\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

به ازای هر کدام از r ها که در R هستند با سمبل a

$\delta'(R, a) =$ اجتماع همه ← من به کجای میرم

پس تابع transition در DFA مثل NFA است

مجموعه ی q_0 = ابتدای کار

③ $q_0' = \{q_0\}$

④ $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

مجموعه ی state های accept در DFA

accept state = در نهایت کامپیوتر به یک
ختم می شود

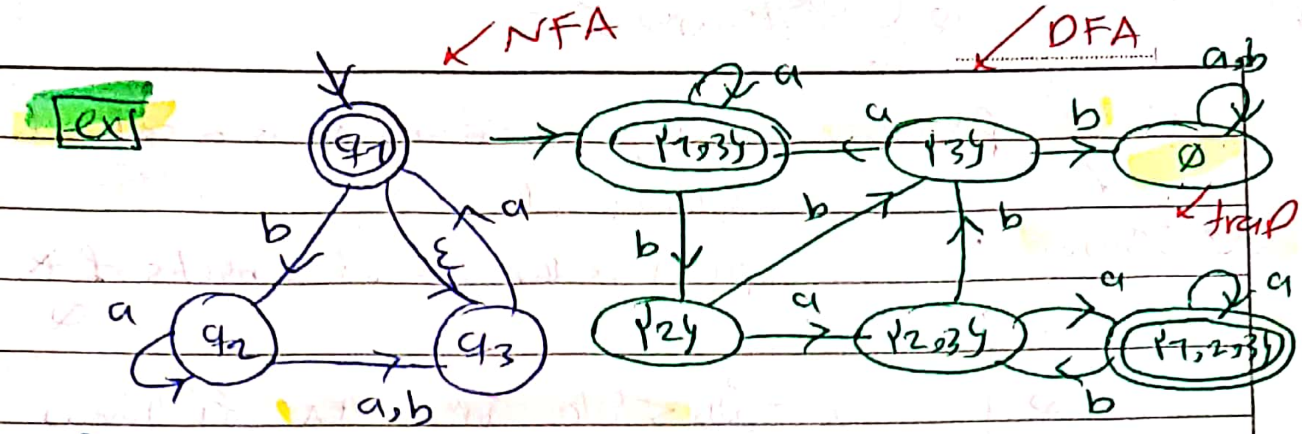
A الان $\boxed{\varepsilon \text{ arrow}}$ اضافه می کنیم در اثبات A

$\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some } r \in R\}$

$$= \bigcup_{r \in R} E(\delta(r, a)).$$

① جای $\delta(r, a)$ را با $E(\delta(r, a))$ عوض کردیم

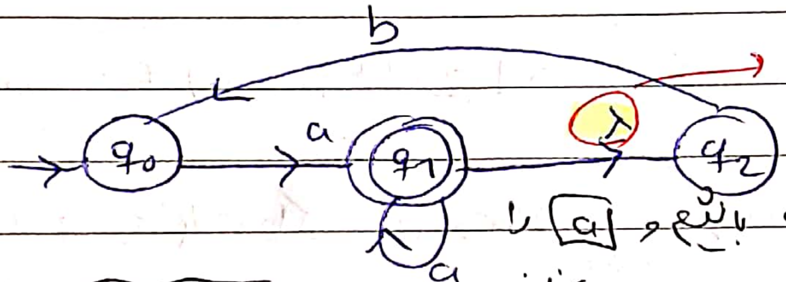
② q_0' را به $E(\{q_0\})$ تغییر می دهیم.



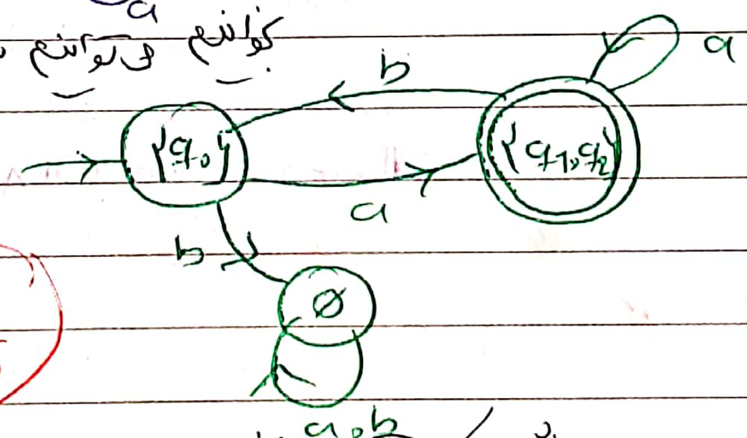
① ابتدای کار یا توی [q1] هستیم یا [q3] ← اگر در q1 یا q3 باشیم و a بخوانیم حتما در مرحله e

$\delta(q_3, a) = q_1$

ابتدا ← ε

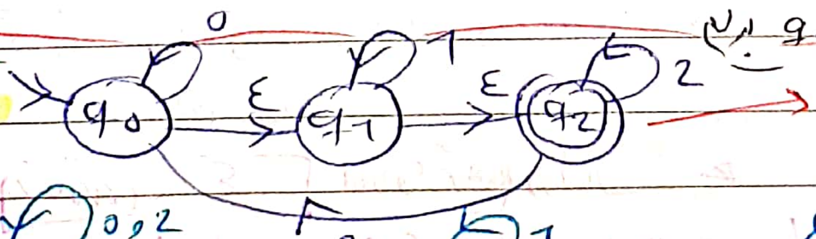


اگر توی q_1 باشیم و [a] را بخوانیم می‌توانیم به (q_1 یا q_2) برویم



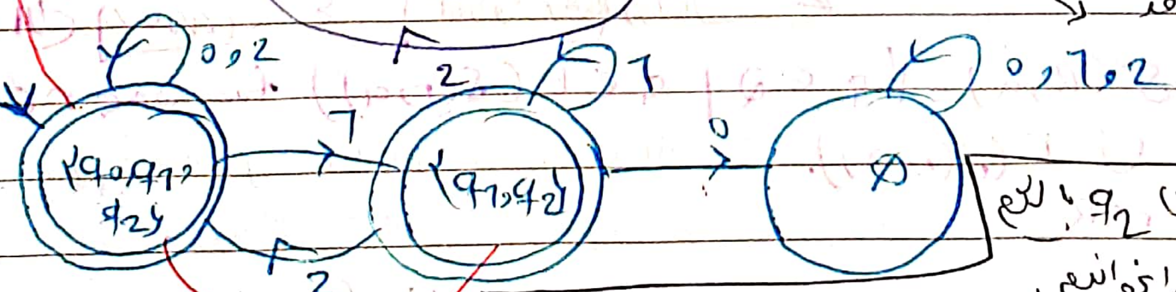
ابتدای کار می‌توانیم به والفبای [ε] توی هر ۲ تا q_1 state باشیم

اگر توی q_0 باشیم و صفر بخوانیم می‌توانیم به stateهای q_0 و q_1 و q_2 برویم



(q_2 = final) به خاطر

ε ها 0, 1, 2

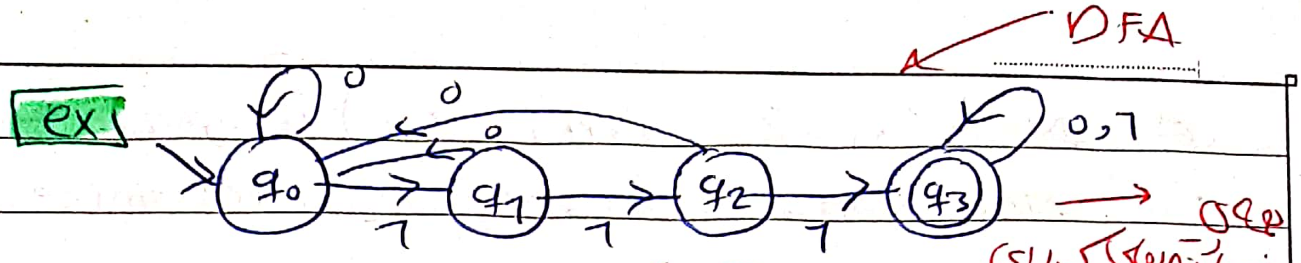


اگر توی q_2 باشیم و 2 را بخوانیم به q_2 یا q_0 نمی‌توانیم برویم که از q_0 هم به q_2 با کاراکتر صفر رسید

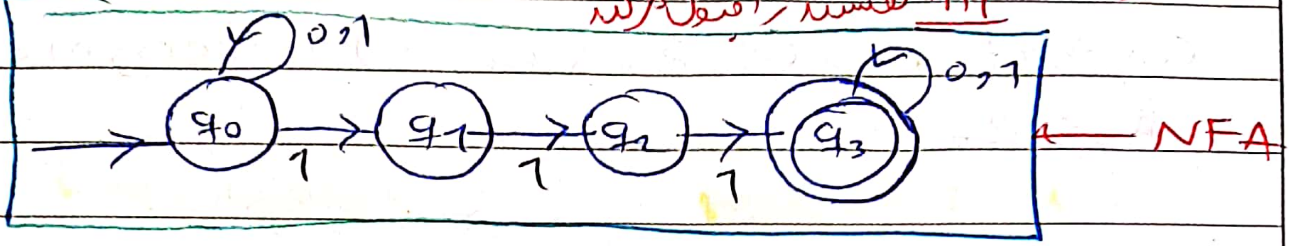
مرکز

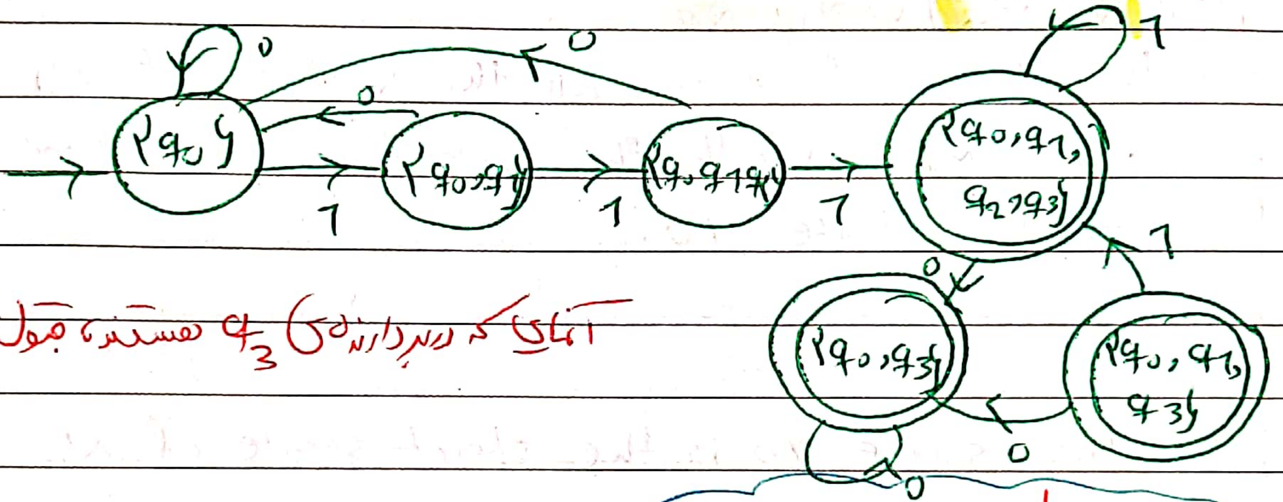
final است که q_2 دارد اضافی

ex:



زیر رشته‌هایی که دارای 111 هستند را قبول می‌کند



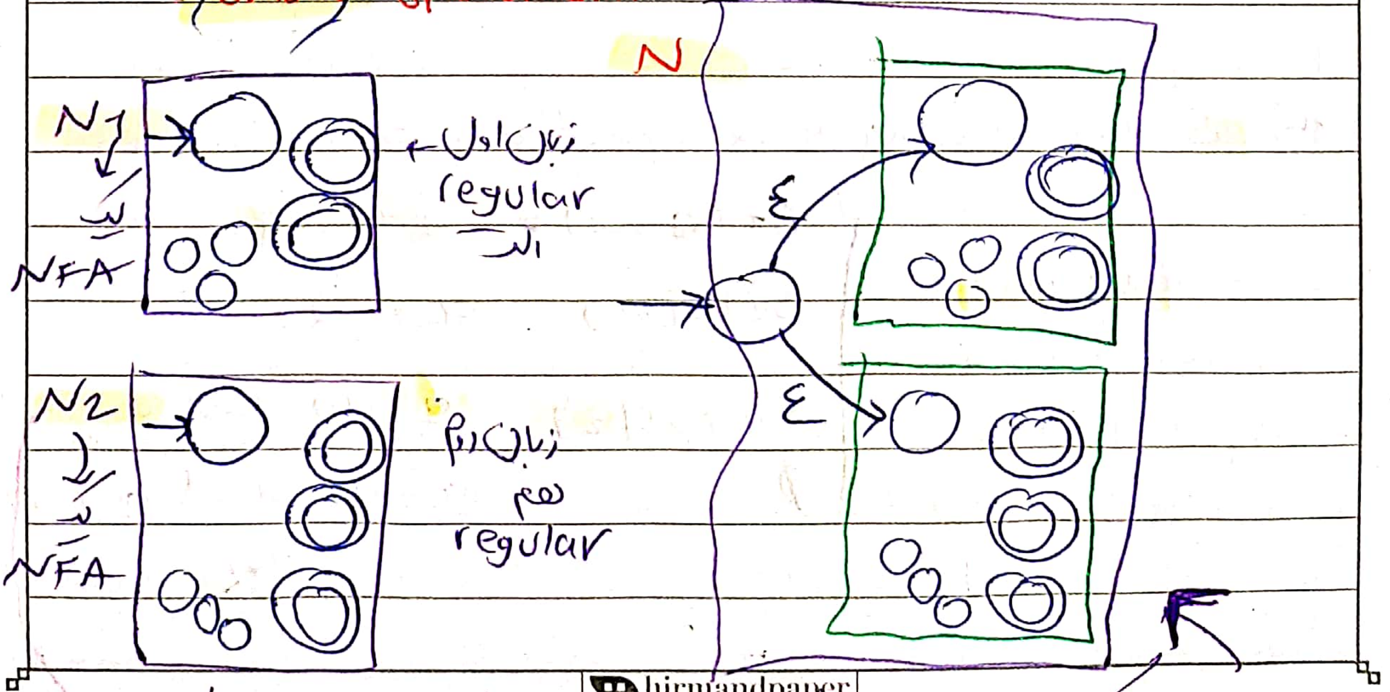


آنهایی که در بردارنده q_3 هستند، قبول می‌کنند

DFA ـ به ـ تبدیل ـ NFA ـ بارا

اجتماع

the class of regular languages is closed under the union operation.



می‌خواهیم یک NFA داشته باشیم که زبان تبدیل آن شامل اجتماع زبان تبدیل N_1 و N_2

let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$

افزای اپسیلون ε — یک start جدید

① $Q = \{q_0\} \cup Q_1 \cup Q_2$

the states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.

② the state q_0 is the start state of N

③ the set of accept states $F = F_1 \cup F_2$

the accept states of N are all the accept states of N_1 and N_2. that way, N accepts if either N_1 accepts or N_2 accepts.

④ δ for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

برای state های که داخل $\cancel{Q_1}$ یا Q_1 هستند

انتقال‌شون همون انتقال قبلی

الد

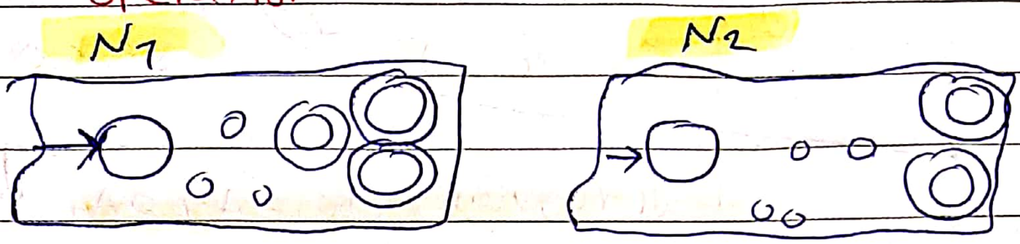
(فقط) انتقال قبلی

چیزی غیر از $\boxed{\varepsilon}$

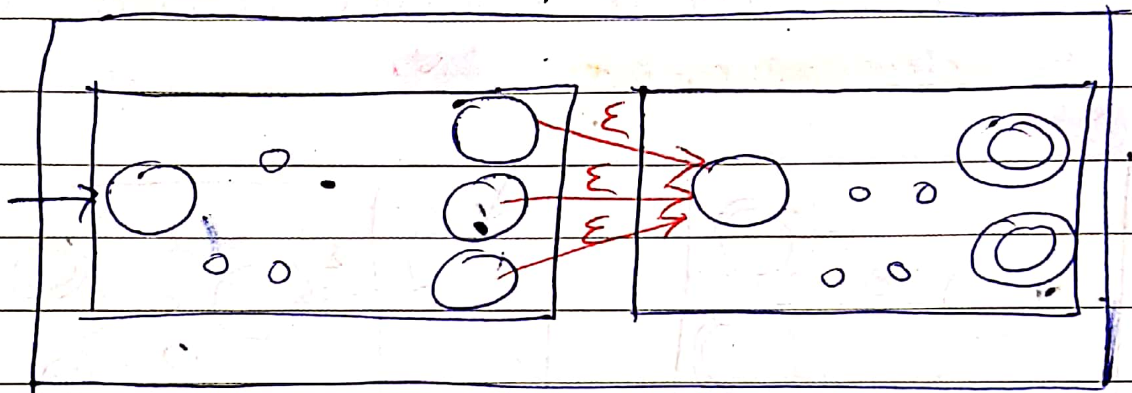
به ازای چیزی غیر از ε خروجی نداریم (تهی میشود)

hirmandpaper

The class of regular language is closed under the concatenation operation



فاینال‌های N = فاینال‌های N2



رشته‌هایی که قسمت اولشون در N1 اند، قسمت دوم هم توسط N2 ، accept میشه. (باید توسط N1، N2 قبول بشوند)

مجموعه فاینال‌های N، فاینال N2 اند

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ " A_2

$N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

① $Q = Q_1 \cup Q_2$ start state

states of N are all the states of N_1 and N_2

② the state q_1 is the same as the start state of N_1.

③ the accept states F_2 are the same as the accept states of N_2.

④ δ for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

تابع ε فضای ابتدای

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

اگر $A_1 \circ A_2$ زبان regular باشد

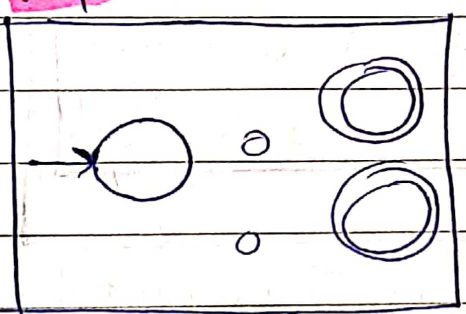
start state

آغاز = state

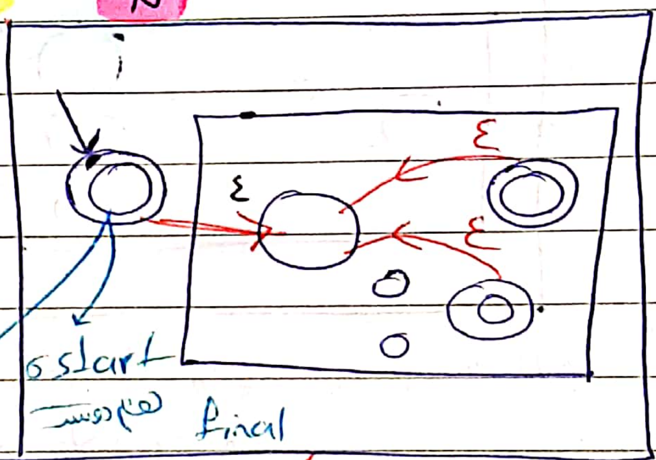
N

the class of regular languages is closed under the star operation

N_1 .



N



این را اضافه کردیم تا ε بپذیرد

باید برای A_1^* یک NFA داریم (برای اثبات منظم بودن) ثابت کنیم

از صفر تا ... بار چسباندن یک رشته به خودش و این عکس کار می کند

A ستاره ی هر زبانی که در بر دارنده ی ε هم هست ← صفر بار رشته های این

زبان اصلی را به هم همواره بچسبانید که $\boxed{\varepsilon}$ به دست میاد .

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.

$N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

① $Q = \{q_0\} \cup Q_1$ the states of N are the states of N_1 plus a new start state

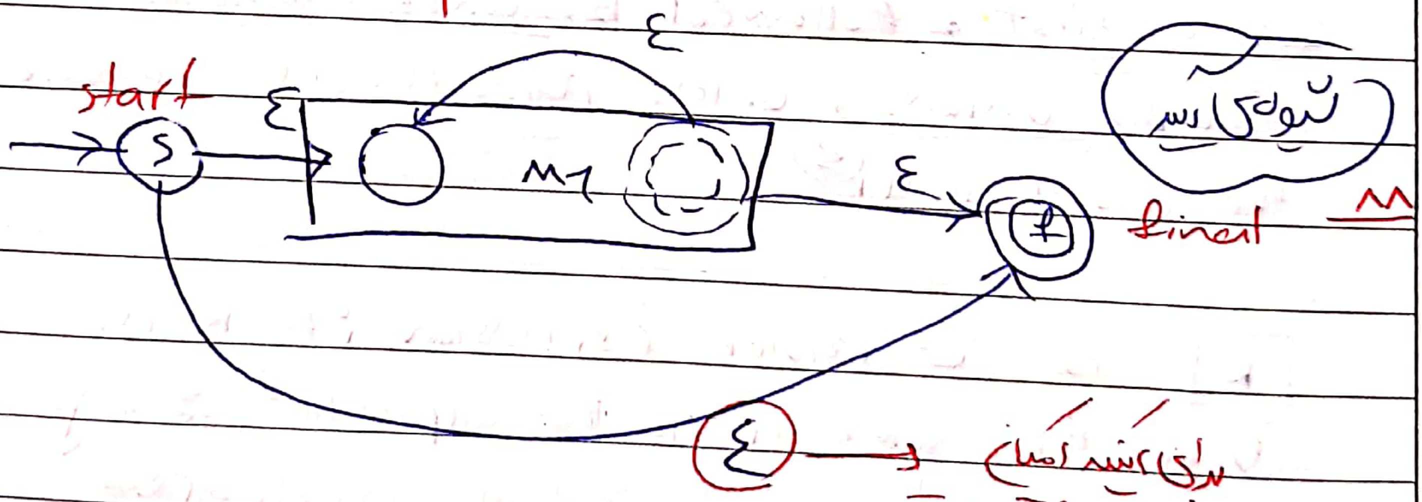
② the state q_0 is the new start state

③ $F = \{q_0\} \cup F_1$

the accept states are the old accepts states plus the new start state.

④ $\boxed{\delta}$ for any $q \in Q$ and $a \in \Sigma_\varepsilon$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1,\ q \notin F_1 \\ \delta_1(q,a) & q \in F_1,\ a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1,\ a = \varepsilon \\ \{q_1\} & q = q_0,\ a = \varepsilon \\ \phi & q = q_0,\ a \neq \varepsilon \end{cases}$$



برای اینکه امکان
پذیرش رشته‌ی تهی
فراهم شود

* عبارت منظم ← یک زبانه *

« Regular expressions »

this notation involves a combination of strings of symbols from some alphabet Σ, parantheses, and the operators $\cup$, $\circ$ and $*$

we can use the regular ~~expressions~~ operations to buildup expressions describing languages, which are called regular expressions. ex: $(0 \cup 1)0^*$

the value of a regular expression is a language

« تقدم و تاخر عملگرها در عبارات منظم »

in regular expressions the star (1) operation is done first, followed by concatenation (2) and finally union (3), unless parantheses change the usual order

$\boxed{R}$ is a regular expression if R is

1. a for some a in the alphabet Σ,
2. ε, هر کدام از نماد های الفبای Σ یک عبارت منظم اند
3. $\emptyset$, تهی
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressons.
5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions, or
6. (R_1^*) where R_1 is a regular expression

the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$. regular expression ϕ represents the empty language

بجای اجتماع ($\cup$) می‌شه (+) هم گذاشت

〃 concat و نقطه (.) هم می‌گذارند.

* $\emptyset, \lambda, a \in \Sigma$ are all regular expressions.
these are called primitive regular expressions.

* A string is a regular expression if and only if it can be derived from the primitive regular expressions by a finite number of applications

زبان تولید عبارت منظم R را با $L(R)$ نشان می‌دهیم.

«(L) زبان تولید یک عبارت منظم»

the language $L(r)$ denoted by any regular expression r is defined by the following rules.

1. $\emptyset$ is a regular expression denoting the empty set

زبان تولید λ

2. λ is a regular expression denoting $\{\lambda\}$
3. for every $a \in \Sigma$, a is a regular expression denoting $\{a\}$

$L(\phi) = \emptyset$

if r_1 and r_2 are regular expressions then

[illegible] 4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$

$L(a) = \{a\}$

اتصال 5. $L(r_1 \cdot r_2) = L(r_1) L(r_2)$

$L(\lambda) = \{\lambda\}$

6. $L((r_1)) = L(r_1)$, 7. $L(r_1^*) = (L(r_1))^*$

$a^* \cup b^*$ → رشته‌های فقط a یا فقط b

ex) $(0 \cup 1)^*$ → it starts with the language $(0 \cup 1)$ and applies the $\boxed{*}$ operation.

مانند یک

the value of this expression is the language consisting of all possible strings of 0s and 1s

همه‌ی رشته‌های متشکل از صفر و 1

کل رشته‌هایی که از 0 و 1 درست کرده‌اند

if $\Sigma = \{0, 1\}$, we can write Σ as shorthand for the regular expression $(0 \cup 1)$.

if Σ is any alphabet, the regular expression Σ describes the language consisting of all strings of length 1 over this alphabet, and Σ^* describes the language consisting of all strings over that alphabet.

ex) $(\Sigma^* 1)$ is the language that contains all strings that end in a 1.

ex) $(0\Sigma^*) \cup (\Sigma^* 1)$ consists of all strings that start with a 0 $\boxed{\text{or}}$ end with a $\boxed{1}$

((ضرورت دقت هنگام مواجهه با عبارات منظم))

① $\boxed{a^* \cup b^* \neq (a \cup b)^*}$

زبان سمت راست حاوی رشته‌های ab است درحالی که زبان سمت چپ حاوی فقط a ها

زبان چپ نه

یا فقط b ها است.

و زبان راست نه.

② $(ab)^* \neq a^* b^*$

زبان سمت چپ حاوی رشته‌ی abab است درحالی که زبان سمت راست

حاوی رشته‌ی aaabbbbbb است درحالی که زبان سمت چپ حاوی این

رشته نیست.

if we let R be any regular expression, we have the following identities.

① $R \cup \emptyset = R$ → Adding the empty language to any other language will not change it.

② $R \circ \varepsilon = R$ joining the empty strings to any string will not change it.

③ $R \cup \varepsilon$ may not equal R. if $R = 0$, then $L(R) = \{0\}$ but $L(R \cup \varepsilon) = \{0, \varepsilon\}$.

④ $R \circ \emptyset$ may not equal R. if $R = 0$, then $L(R) = \{0\}$ but $L(R \circ \emptyset) = \emptyset$.

$R \circ \emptyset = \emptyset$ → برای هر R

R^*, R^+

we let R^+ be shorthand for RR^*.
whereas R^* has all strings that are 0 or more concatenations of strings from R,
the language R^+ has all strings that are 1 or more concatenations of strings from R.
→ $R^+ \cup \varepsilon = R^*$ we let R^k be shorthand for the concatenation of k R's with each other.

R^* → صفر یا بیشتر

R^+ → یک یا بیشتر

$\boxed{R \text{ تا } ۱۰ \text{ یعنی } R^{10}}$

examples: $\Sigma = \{0, 1\}$

حتماً یک (1) دارد.

① $0^*10^* = \{w \mid w$ contains a single $1\}$

حداقل یک (1) دارد.

② $\Sigma^* 1 \Sigma^* = \{w \mid w$ has at least one $1\}$

③ $\Sigma^* 001 \Sigma^* = \{w \mid w$ contains the string 001 as a substring$\}$

دوتا صفر پشت سرهم نداریم →

④ $1^*(01^+)^* = \{w \mid$ every 0 in w is followed by at least one $1\}$

همه رشته‌های باینری به طول زوج

⑤ $(\Sigma\Sigma)^* = \{w \mid w$ is a string of even length$\}$

رشته‌های به طول ۱ تا n ضرب ۳ است

⑥ $(\Sigma\Sigma\Sigma)^* = \{w \mid$ the length of w is a multiple of $3\}$

سمبل‌های ابتدا و انتهای رشته یکی است

⑦ $01 \cup 10 = \{01, 10\}$

⑧ $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w$ starts and ends with the same symbol$\}$

⑨ $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$

the expression $(0 \cup \varepsilon)$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.

⑩ $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$

⑪ $1^*\emptyset = \emptyset$ concatenating the empty set to any set yields the empty set.

(12) $\emptyset^* = \{\varepsilon\}$

The star operation puts together any number of strings from the language to get a string in the result. if the language is empty, the star operation can put together 0 strings, giving only the empty string.

ex) $(0+1)^* 110 (0+1)^* = \{w \in \{0,1\}^* \mid$ w includes 110 as a substring $\}$

Concate

ex) $(0+1)(0+1)(0+1)(0+1)^* = \{w \in \{0,1\}^* \mid$ the length of w is greater than or equal to $3\}$

ex) $0+1+0(0+1)^*0+1(0+1)^*1 = \{w \in \{0,1\}^* \mid$ the first and the last symbols of w are the same$\}$.

ex) $0^*1^*2^* = \{0^i 1^j 2^k \mid i \geq 0, j \geq 0, k \geq 0\}$

Regular expressions are useful tools in the design of compilers for programming languages. for example, a numerical constant that may include a fractional part and/or a sign may be described as a member of the language.

hirmandpaper

ε یا - یا +

حداقل یک کد

$(+ \cup - \cup \varepsilon)\ (D^+ \cup D^+ . D^* \cup D^* . D^+)$ → ③ ②

sign — علامت — concat

بخش اعشاری کد

توصیف می کنیم کد های اعشاری را با یک عبارت منظم توصیف کنیم.

چهار ابزار اعشاری را با ان ها

می توانیم نشان دهیم

② حالت صحیح می تواند کد نباشد / سمت چپ صحیح حداقل یک کد باید باشد

③ سمت چپ صحیح کدی می تواند نباشد اما حالت باید حداقل یک کد داشته باشد.

D = {0, 1, 2, ..., 9} is the alphabet of decimal digits.

examples: 72, 3.14159, +7., -.07.

صحیح — صحیح

ex) the language of strings in $\{a, b\}^*$ with an odd number of a's

فرد a — غلط

جواب اشتباه: $b^*ab^*(ab^*a)^*b^*$ ✗ is incorrect, because it doesn't allow b's between the second $\boxed{a}$ in one of the repeating pairs ab^*a and the first a in the next pair. one correct regular expression describing the language is

$b^*ab^*(ab^*ab^*)^*$ ✓ درست

غلط ✗

the expression $b^*a(b^*ab^*ab^*)^*$ is also not correct, because it doesn't allow strings with just one $\boxed{a}$ to end with $\boxed{b}$, and the expression $(b^*a(b^*ab^*a)^*b^*)$ corrects the mistake.

درست

Another correct expression: $(b^*a(b+ab^*a)^*)$ درست

All of these could also be written with the single $\boxed{a}$ on the right, as in

$(b+ab^*a)^*ab^*$

درست

$\boxed{ex}$ the langrage of strings in $\{a,b\}^*$ Ending with $\boxed{b}$ and Not containing $\boxed{aa}$

$(b+ab)^*(b+ab)$

پس از هر a، بلافاصله b داریم

$(b \cup ab)^+$ → حداقل یک بار ظاهر شود

به عبارت دیگر

چرا $(b+ab)^*$ نمی‌تواند جواب مسئله باشد؟

چون می‌تواند زبان تهی را توصیف کند که با b خاتمه پیدا نمی‌کند.

$\boxed{ex}$ given a language, find a regular expression

Let $L=\{w\in\{a,b\}^* : |w| \text{ is even}\}$

regex: $((a\cup b)(a\cup b))^*$

this can be read as "Go through a loop zero or more times. each time through, choose an a or b then choose a second character (a ~~or~~ or b)" regex 2: $(aa \cup ab \cup ba \cup bb)^*$

this can be read as "Go through a loop zero or more times, each time through, choose one of the two character sequences"

ex «More than one regex for a language»

let $L = \{w \in \{a,b\}^* : w$ contains an odd number of a's$\}$

مجموعه رشته‌های باینری که تعداد a های آن‌ها فرد است

two equally simple regular expressions that define L are:

a 2k یک دانه

1 روش $b^*(ab^*ab^*)^*a\ b^*$

2 روش $b^*ab^*(ab^*ab^*)^*$

(a اضافه) یک دانه

* یک زبان منظم را با بی‌نهایت regex می‌توان توصیف کرد

ex for $\Sigma = \{0,1\}$ give a regex r such that $L(r) = \{w \in \Sigma^* : \boxed{w}$ has at least one pair of consecutive zeros$\}$.

رشته‌ای که حداقل یک $\boxed{00}$ داشته باشد

$r = (0+1)^*00(0+1)^*$

complement مثال قبلی (نقیض مثال قبل)

ex find a regex for

$L = \{w \in \{0,1\}^* : w$ has no pair of consecutive zeros$\}$.

رشته‌ای که دو تا صفر پشت سر هم نداشته باشد.

$r = (1^*011^*)^*(0+\varepsilon) + 1^*(0+\varepsilon)$

برای اینکه رشته‌ی ما کدام یک هم تو انتهای رشته باشیم.

برای اینکه صفر تنها این در انتهای رشته حضور داشته باشد

→ concate شده

$r = (1+01)^*(0+\varepsilon)$ راه‌کسر

★ Generally, there are an unlimited number of regex for any given language ★

Session 8

★ regex and finite automata are equivalent in their descriptive power.

any regex can be converted into a finite automaton that recognize the language it describes.

Recall that a regex is one that is recognized by some finite automaton.

⇓

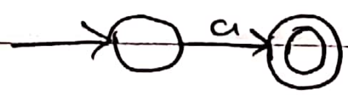
for every regex there is a regular language
And
for every regular language there is a regex.

if and only if

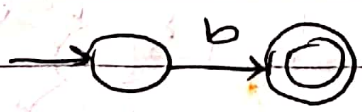
اثبات اینکه تقلیل هر عبارت منظم یک NFA داریم ؟ به کتاب مراجعه کن

ex

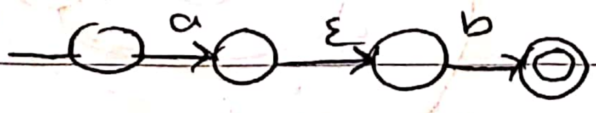
① a



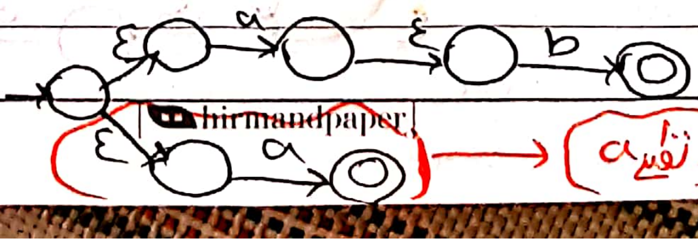
② b



③ ab



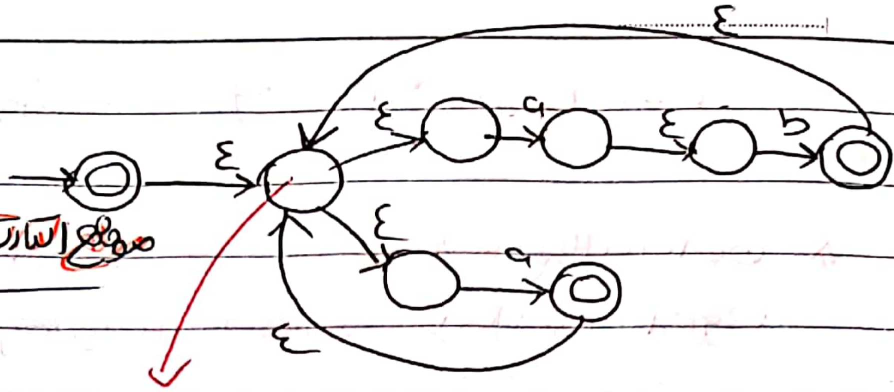
④ ab ∪ a



→ a تقلیل_NFA

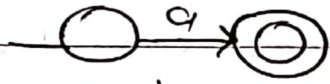
Building an NFA from the regex $(ab \cup a)^*$

⑤ $(ab \cup a)^*$

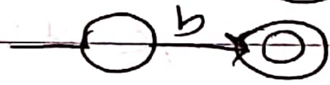


ex Building an NFA from regex $(a \cup b)^* aba$

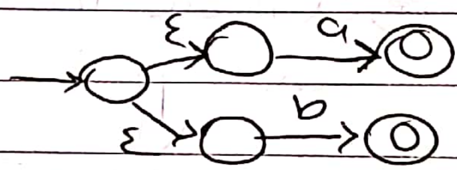
① a



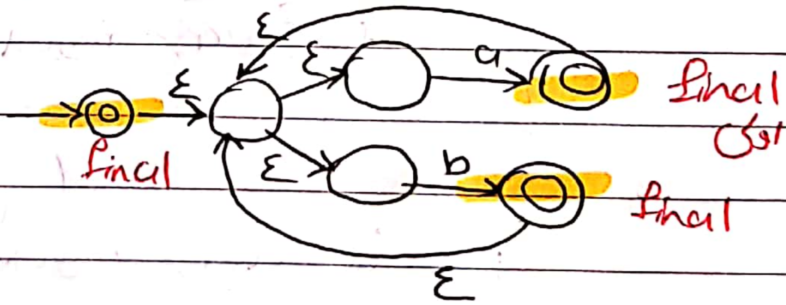
② b



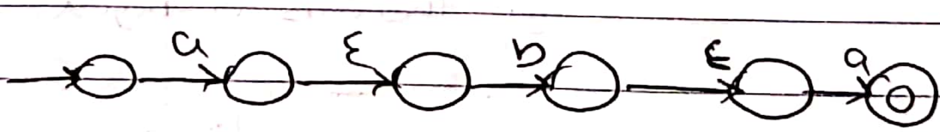
③ $a \cup b$



④ $(a \cup b)^*$



⑤ aba



⑥ $(a \cup b)^*$ aba

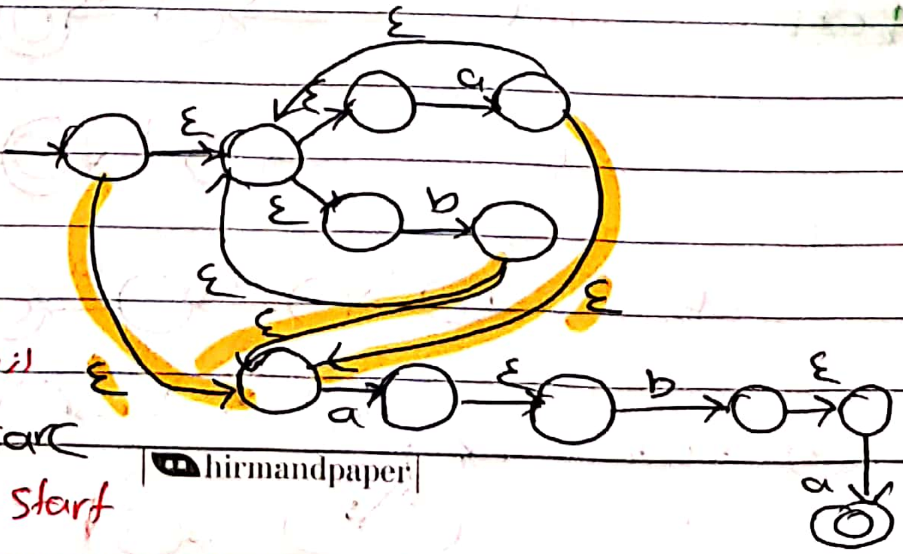
برای concat کن

↓

ε از موقعیت فاینال اولی یک

arc ترسیم میکنیم به موقعیت

start = دوم



[ex] convert the regex to an NFA:

$(abab)^* \cup (aaa^* \cup b)^*$

Lemma: if a language is regular, then it is described by a regex

* we need to show that if a language $\boxed{A}$ is regular, a regular expression describes it.
Because A is regular, it is accepted by a DFA.

we describe a procedure for converting DFAs into equivalent regex.

we break this procedure into two parts, using a new type of finite automaton called: generalized nondeterministic finite automaton

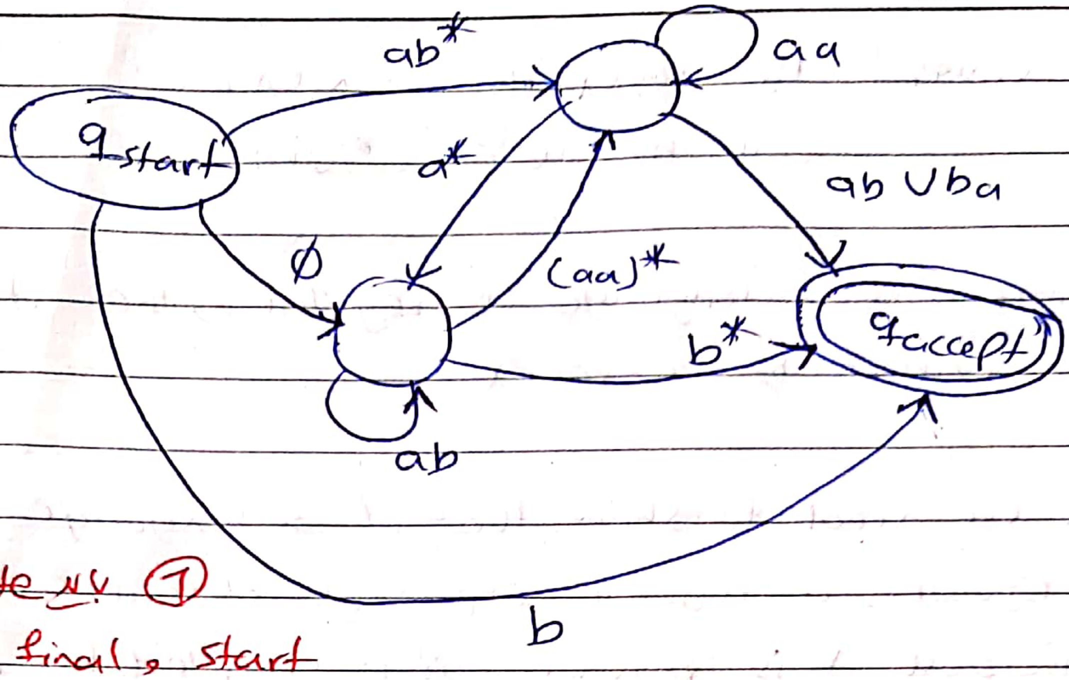
GNFA

① GNFA are nondeterministic finite automata wherein the transition arrows may have any regex as labels, instead of only members of the alphabet or ε.

* GNFA علاقتها بلاها إياش غولنا

② reads block of symbols from input, not just one symbol at a time as in an ordinary NFA

③ A GNFA is nondeterministic and so many have different ways to process the same input string



① باید state های start و final یه واحد داشته باشن

② به start یالی وارد نشود

③ از accept یالی خارج نشود

④ از هر رأس به رأس دیگر یال داشته باشیم

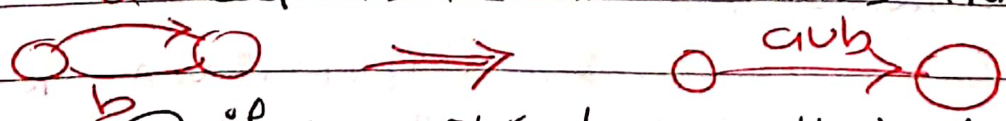
① تبدیل DFA به GNFA

② GNFA به یک regex معادل

* از start به همه state های دیگر arrow داریم.

How to ~~add~~ convert a DFA into a GNFA in the special form.

① we simply add a new start state with an ε arrow to the old start state and a new accept state with ε arrows from the old accept states



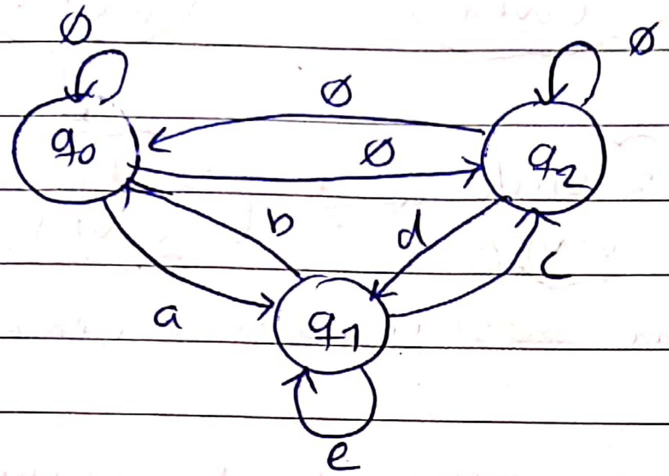
② if any arrows have multiple labels (or if there are multiple arrows going between the same two states in the same direction), we replace each with a single arrow whose label is the union of the previous labels.

توضیح

اجتماع می گیریم

* اضافه کردن label ∅ بین دوتا state که یال وجود نداره

③ we add arrows labeled ∅ between states that had no arrows. this step won't change the language recognized because a transition labeled with ∅ can never be used.



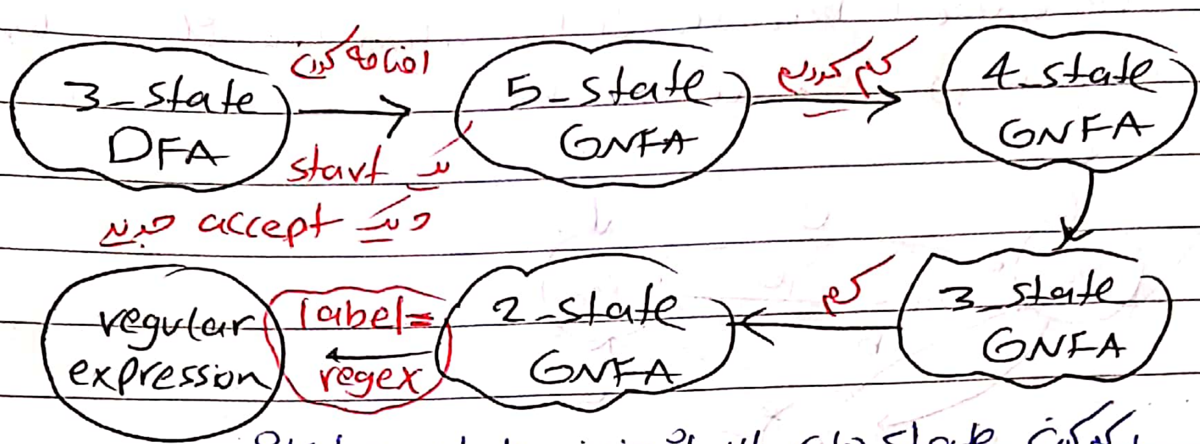
how to convert a GNFA to a regex?

① if GNFA has K states then because a GNFA must have a start and an accept state and they must be different from each other → $K \geq 2$

② if $K > 2$, we construct an equivalent GNFA with K−1 states

③ if $K = 2$, the GNFA has a single arrow that goes from the start state to the accept state. the label of this arrow is the equivalent regex.

the stages in converting a DFA with three states to an equivalent regex :



با کم کردن state ها باید اثر حذف state را جبران کنیم

the crucial step is constructing an equivalent GNFA with one fewer state when $K > 2$.

تا می کشیم   بیرون می اندازیم

* we do so by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized. باز هم ماند

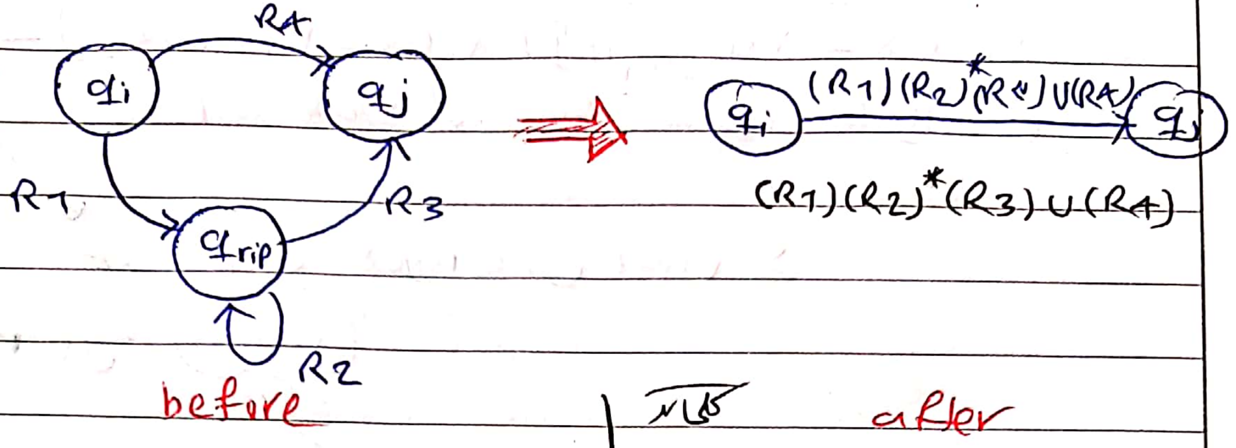
تا اثر حذف و جبران کنیم

* Any state will do, provided that it is not the start or accept state. we are guaranteed that such a state will exist because $K > 2$. اون state که انتخاب می کنیم تا حذف کنیم باید غیر از start , accept باشد

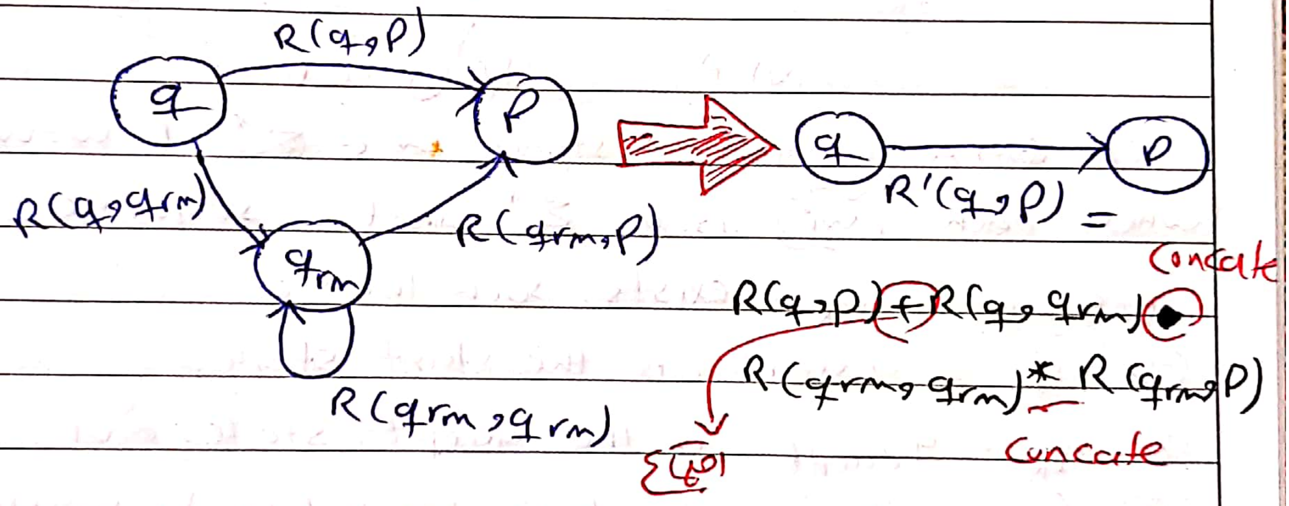
* lets call the removed state qrip. After removing qrip we repair the machine by altering the regex that label each of the remaining arrows. q' state که حذف می کنیم
The new labels compensate for the absence of qrip by adding back the lost computations.

ex

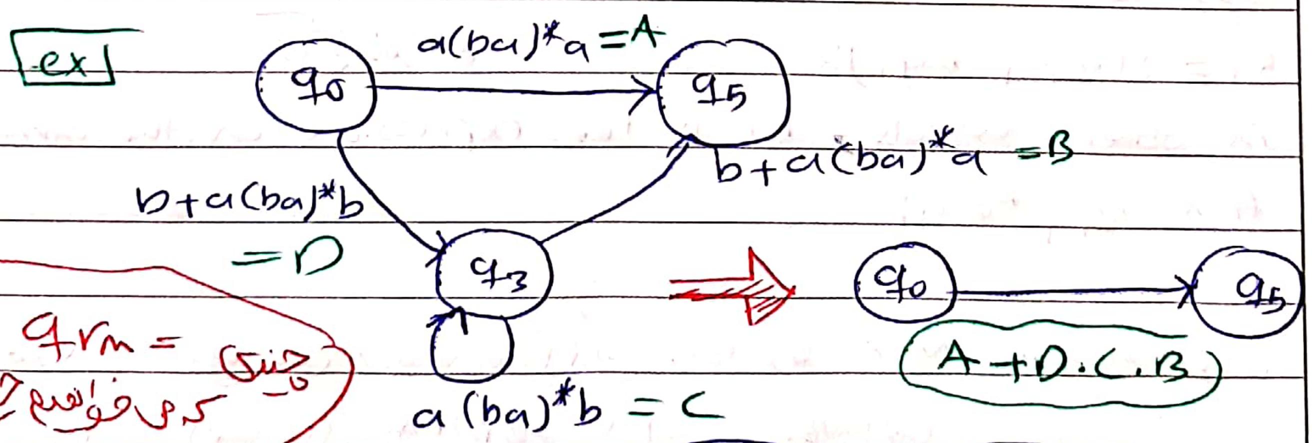
Constructing an equivalent GNFA with one fewer state.

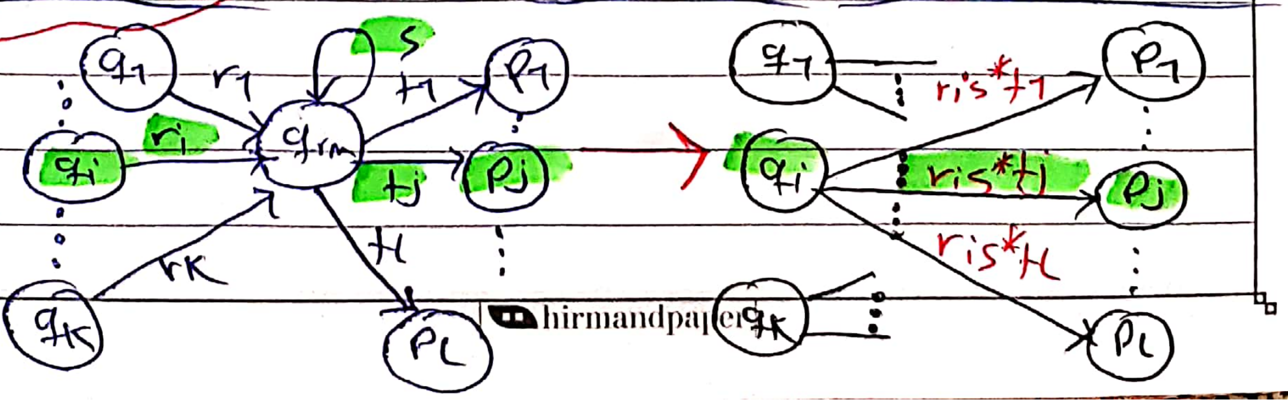


ex



ex





تابع انتقال یک GNFA به شکل زیر: formally:

state ـ پایان — state ـ آغاز

$\delta: (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R$

از هر state به state دیگر دقیقاً یک یال داریم

مجموعه‌ی همه‌ی regex ها روی الفبای Σ

R (label) ها

تمرین: تابع انتقال یک GNFA را با تابع انتقال یک DFA و NFA مقایسه کنید

مفهوم پذیرش رشته برای یک GNFA

A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1 w_2 \ldots w_k$ where each w_i is in Σ^* and a sequence $q_0, q_1, \ldots, q_k$ exists such that:

1. $q_0 = q_{start}$ is the start state.
2. $q_k = q_{accept}$ is the accept state, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$;

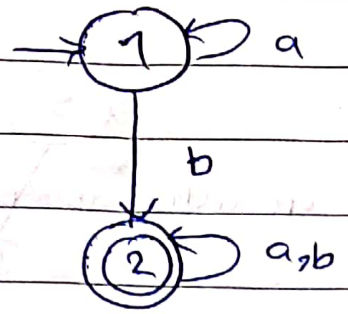
زبان تفسیر R_i

in other words, R_i is the expression on the arrow from q_{i-1} to q_i.

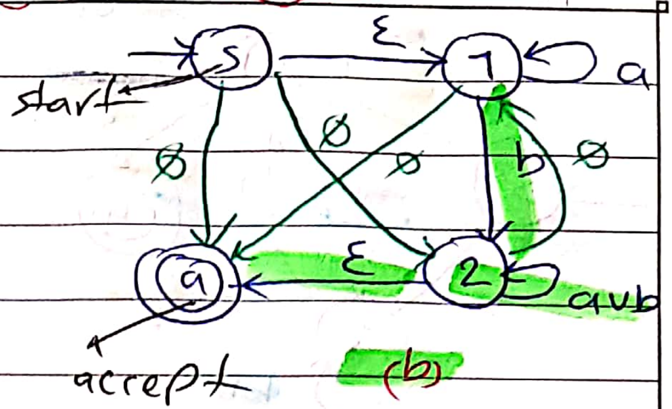
تمرین: تعریف رسمی بالا برای پذیرش یک رشته توسط GNFA را با تعریف رسمی مفهوم پذیرش برای DFA ها و NFA ها مقایسه کنید

DFA تبدیل به NFA

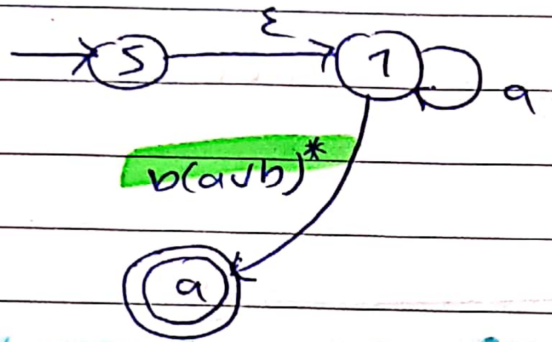
ex



(a)



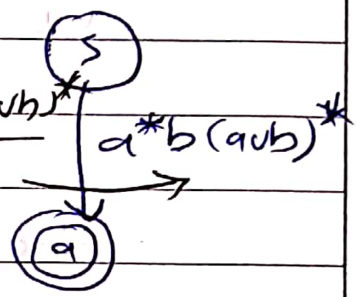
(b)



(c)

$\underline{\varepsilon . a^* . (b)(a\cup b)^*}$ ⟶ $a^*b(a\cup b)^*$

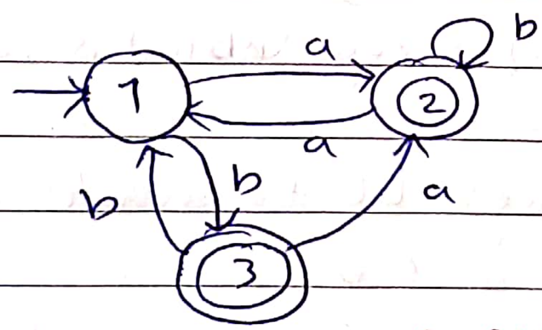
یعنی

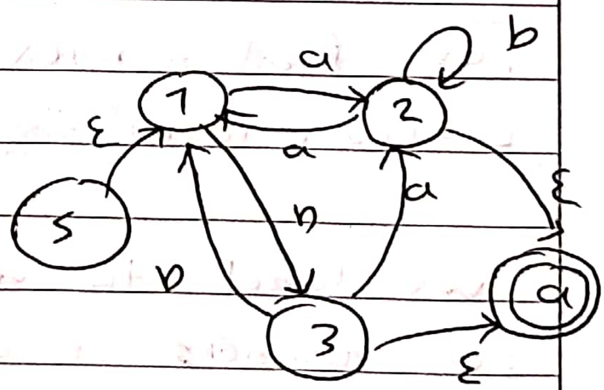


(d)

حذف 2 state ● حذف 1 state

ex





(arc ها با label ∅ رسم نشدن :))

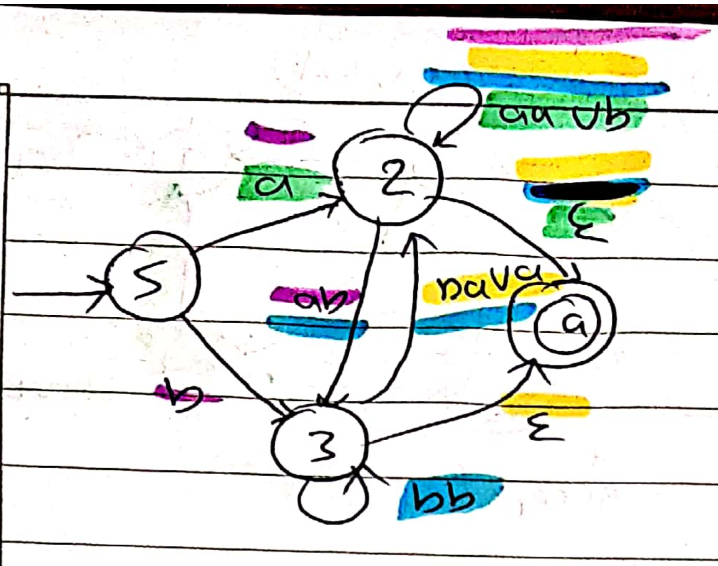
(a) (b)

$\emptyset^* \Rightarrow \varepsilon$

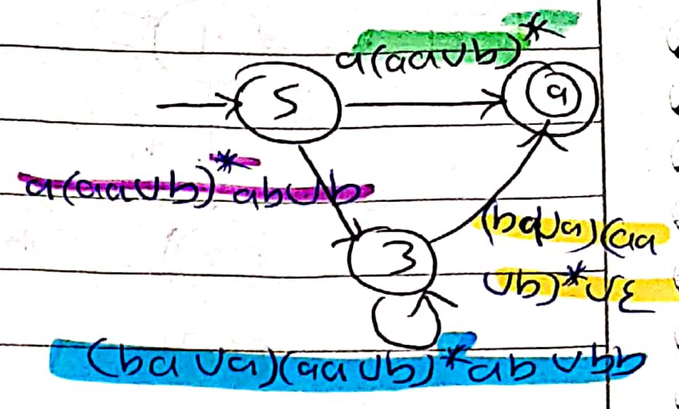
مهم

S → 2
S → 3
2 → 2
3 → 3
2 → 3
3 → 2

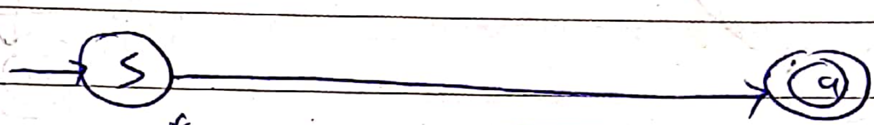
این arc ها با ... اصلاح شود



(c)



(d)



$(a(aa\cup b)^*ab\cup b)((ba\cup a)(aa\cup b)^*ab\cup bb)^*$

$((ba\cup a)(aa\cup b)^*\cup\varepsilon)\cup a(aa\cup b)^*$

$\boxed{ex}$ find a regex for language

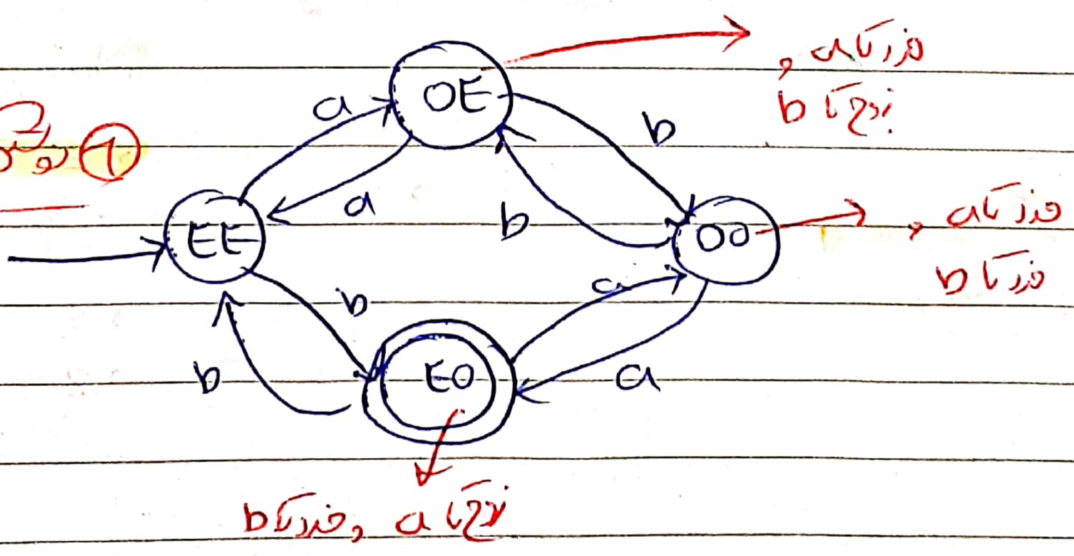
$L=\{w\in\{a,b\}^* : n_a(w)$ is even $n_b(w)$ is odd$\}$.

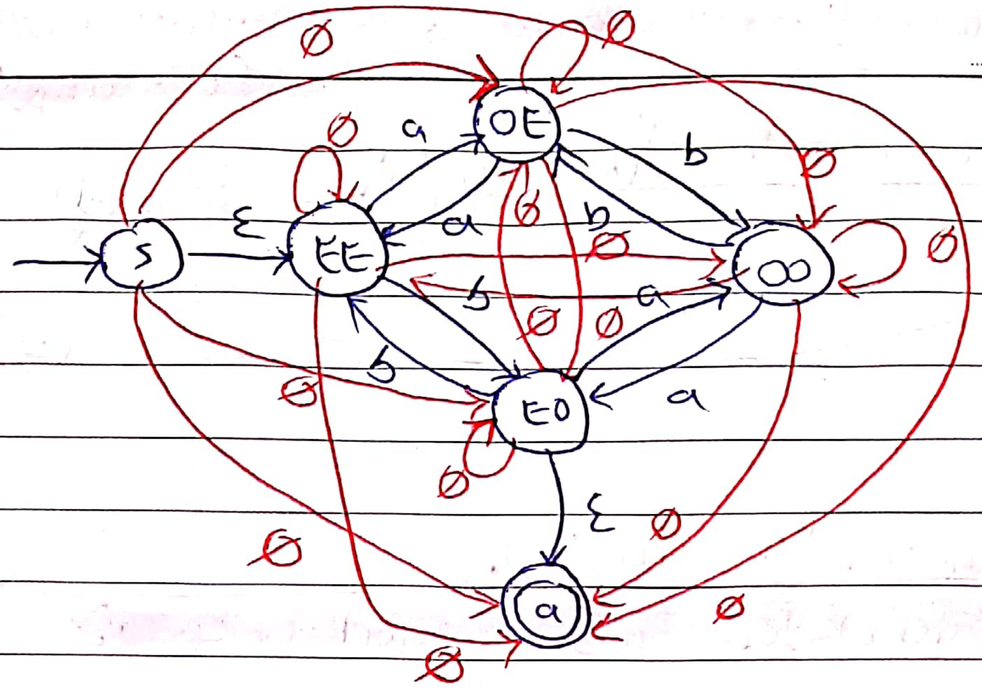
* مثال مهم *

we label the vertices EE to denote an even numbers of a's and b's.

OE → odd number of a's & even number of b's

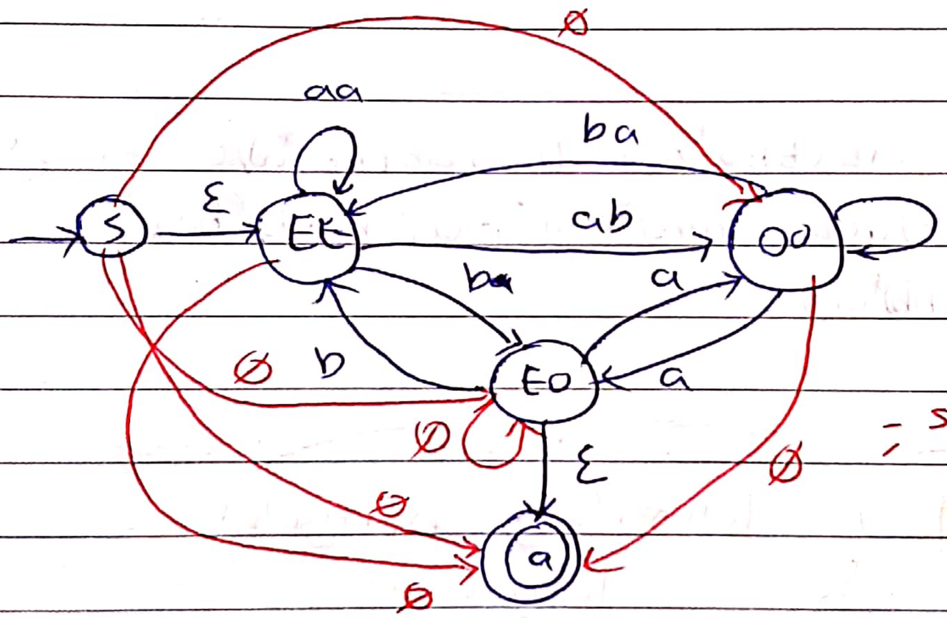
DFA رسم (1)





2

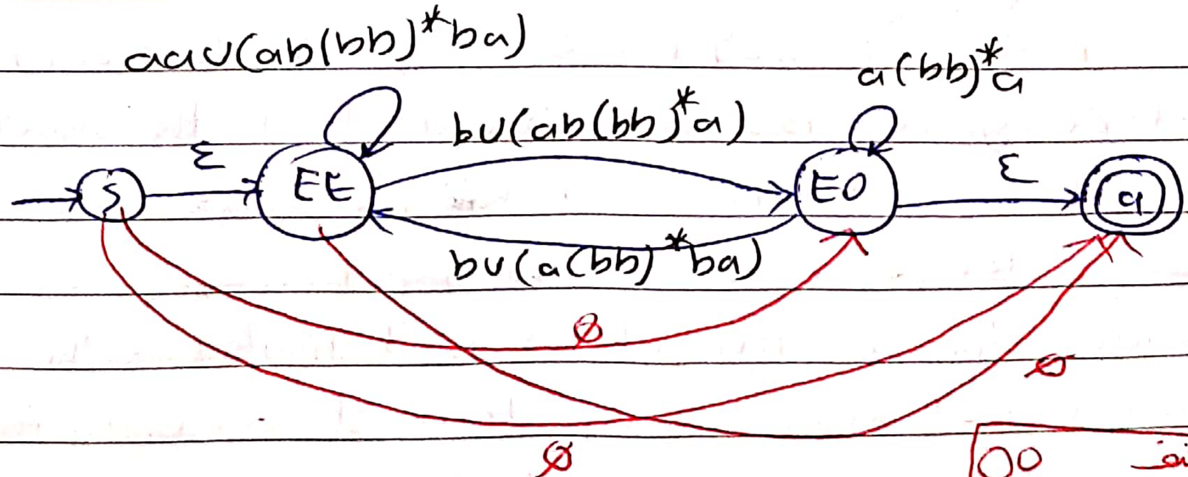
تبدیل به GNFA



3

حذف state -

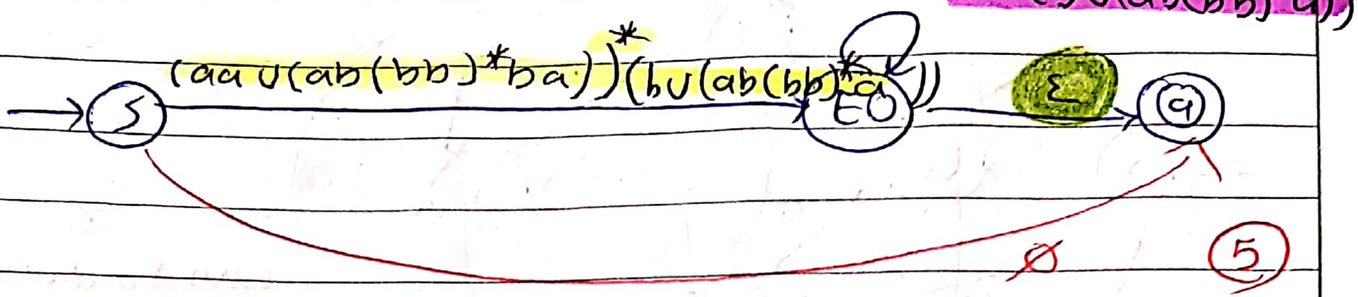
OE



4

حذف OO

$(a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^*$
$(b \cup (ab(bb)^*a))$



(5)

$\rightarrow (s) \xrightarrow{(aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a)) \cdot ((a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^* \cdot (b \cup (ab(bb)^*a))} (q)$

(6)

$R = (aa \cup (ab(bb)^*ba))^*(b \cup (ab(bb)^*a))$ زبان (7)
$((a(bb)^*a) \cup (b \cup (a(bb)^*ba))(aa \cup (ab(bb)^*ba))^* \cdot$
$(b \cup (ab(bb)^*a))$.

session 8

Identifying nonregular languages

نامنظم

لغوی توانند حافظه کننده باشند. اثبات برای لازم است.

مثال $B = \{0^n 1^n \mid n \geq 0\}$ if we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0's have been seen so far as it reads the input.

Because the number of 0's isn't limited $\rightarrow$ the machine will have to keep track of an unlimited number of possibilities. But it can't do so with any finite number of states.

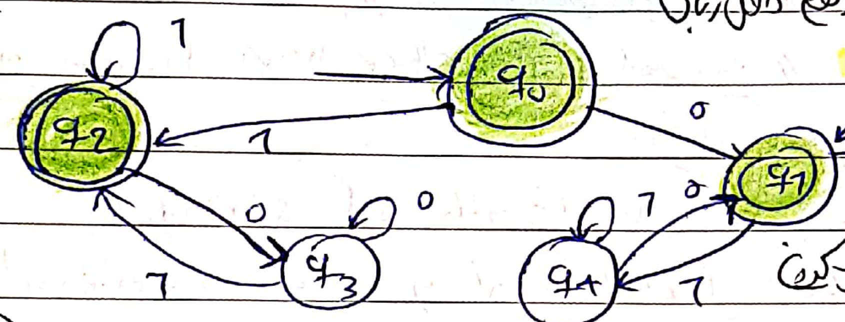
$\Sigma = \{0,1\}$

ex) $C = \{w \mid w$ has an equal number of 0s and 1s$\}$ → is $\boxed{\text{not regular}}$

$D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings$\}$ → is regular

که می‌توانیم این قسمت میانی را
تکرار کنیم و این رشته‌ای که D تعریف DFA
حاصل می‌شود باز هم داخل زبان
می‌ماند

یعنی رشته به یه قسمتی می‌تواند
گیر کند و لوپ



برای این کار می‌گیم پامپ کردن

در رشته‌هایی که طولشان حداقل برابر p است را می‌شود پامپ کرد

the pumping Lemma for Regular languages

اثبات نامنظم بودن یک زبان با لم pumping به کمک برهان خلف

تناقض → contradiction

+ اول فرض می‌کنیم زبان منظم است
سپس به یک تناقض می‌رسیم

رشته‌هایی که داخل یک زبان منظم هستند و طولشان
از یک حدی بزرگتر است را می‌شود پامپ کرد ←

ویژگی مشترک همه‌ی زبان‌های منظم ⇐ طبق تعریف زبان منظم، یک حدی داریم مثل p

all strings in the language can be "pumped" if they are at least as long as a special value called the pumping length.

that means each such string contains a section that can be repeated any number of times with the resulting string remaining in the language

یعنی قسمت میانی را می‌توانیم هی تکرار کنیم بدون اینکه خارج زبان بشود (همچنان جزو زبان هستند) همه‌شان

★ ★ از برقرار نبودن لم تزریق، می‌توانیم منظم نبودن یک زبان را نتیجه بگیریم ★

Pumping lemma: if A is a regular language, then there is a number p (the pumping length) where if s is string in A of length at least p, then s may be divided into $\boxed{3 \text{ pieces}}$

$s = xyz$ satisfying the following conditions:

با تکرار کردن قسمت وسطی ← باید فقط رشته‌هایی که ایجاد می‌شوند خارج از زبان بیفتد

1. for each $i \geq 0$, $xy^iz \in A$ — xyz داخل زبان الف
2. $|y| > 0$ and // $y \neq \varepsilon$
3. $|xy| \leq p$ // x and y together have length at most p

★ $|s|$ represents the length of string s — طول رشته s

y^i means that i copies of y are concatenated together and y^0 equals to ε

when s is divided into xyz, either x or z may be $\boxed{\varepsilon}$ but condition 2 says $y \neq \varepsilon$.

زبان عمومی

زبان ورودی — نامتناهی

بیان دیگری برای لم تزریق

Let A be an infinite regular language then there exists some positive integer p such that $\boxed{\text{any}}$ $s \in A$ with $|s| \geq p$ can be decomposed as $s = xyz$ with $|xy| \leq p$ and $|y| \geq 1$ such that xy^iz is also in A for all $i = 0, \ldots$

every sufficiently long string in $\boxed{A}$ can be broken into three parts in a way that an arbitrary (y) عبارت میانی number of repetions of the middle part yields another string in A → we say that the middle string is "pumped" → hence the term pumping lemma for this result.

★ همه‌ی زبان‌های متناهی ← منظم هستند

* لم تزریق برای هر زبان منظمی برقرار است اما لزومی ندارد اگر لم تزریق برقرار بود، زبان منظم باشد

عکس قضیه درست نیست

نکته مهم: ما حق تعیین تجزیه را نداریم. یعنی حق نداریم که x و y و z چه هستند یا چه نیستند.

ما صرفاً می‌دانیم که یک چنین تجزیه‌ای که ۳ شرط را ارضا کند وجود دارد.
اینکه دقیقاً چیست را نمی‌دانیم. ← استدلال ما نباید مبتنی بر تعیین تجزیه باشد.

اثبات: اینکه لم تزریق برای زبان‌های منظم برقرار است. ?!

*** صورت لم خیلی مهم است ***

proof: let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and $p = |Q|$. let $s = s_1 s_2 \ldots s_n$ be a string in A of length n where $n \geq p$.

let $r_1, r_2, \ldots, r_{n+1}$ be the sequence of states that M enters while processing s. so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. this sequence has length $\boxed{n+1}$ which is at least $\boxed{p+1}$

Among the first $p+1$ elements in the sequence, two must be the same state. by the Pigeonhole principle. ← اصل لانه کبوتری

we call the first of these r_j and the second r_l. Because r_l occurs among the first $p+1$ places in a sequence starting at r_1, we have $l \leq p+1$.

now let $x = s_1 \ldots s_{j-1}$, $y = s_j \ldots s_{l-1}$, $z = s_l \ldots s_n$

از z خارج می‌کنیم و به اولی وارد می‌کنیم

As $\boxed{x}$ takes M from r_1 to r_j, $\boxed{y}$ takes M from r_j to r_j, $\boxed{z}$ takes M from r_j to r_{n+1}, which is an accept state → $\boxed{M}$ must accept xy^iz for $\boxed{i \geq 0}$. we know that $j \neq l$ so $|y| > 0$; and $l \leq p+1$ so $|xy| \leq p$. →

همه‌ی شرط‌های لم تزریق را ارضا کردیم.

examples of pumping lemma

اثبات نامنظم بودن یک زبان به کمک لم تزریق

ex1 B is the $\{0^n1^n | n \geq 0\}$ language

① فرض می‌کنیم B یک زبان منظم است (فرض خلف) ← ② یک $\boxed{P}$ داریم که رشته‌های با طول حداقل P را می‌تواند pump کند

③ اگر $s = 0^P1^P$ ← چون s داخل زبان B است و طولش بیشتر از P است (طول s = 2P) ← s را می‌توان pump کرد و به ۳ قسمت شکست ⟷ ④ $s = xyz$ → for any $i \geq 0$ the xy^iz is in $\boxed{B}$

$|xy| \leq P$

$|y| > 0$

⑤ به خاطر شرط $|xy| \leq P$ است و $|xy| \leq \boxed{P}$ سمبل اول s 0s صفر هستند ⟷ all the symbols in xy must be 0s.

⑥ در نتیجه ← $y = 0^K$ طبق شرط‌های بالا $K > 0$

⑦ طبق شرط یک که می‌گفت به ازای هر i، xy^iz باید داخل زبان باشد در حالی که اینجا xy^iz تعداد $\boxed{\text{صفرها و یک‌ها}}$ با هم فرق می‌کند

مثلاً xy^2z ← $0^{P+K}1^P$ است ← با یک بار pump کردن y باید رشته‌ی حاصل داخل زبان بیفتد اما این طور نشد ← تناقض داریم.

because the pumping lemma says $xy^2z \in L$ but $P+K \neq P$ → این تناقض نشان می‌دهد که فرض ما مبنی بر منظم بودن زبان درست نبوده ← فرض خلف باطل و در نتیجه به تناقض می‌رسیم.

* زبان نامنظم است *

example 2 let $C = \{w \mid w$ has an equal number of 0's and 1's$\}$

(1) Assume to the contrary that C is regular. (2) let $\boxed{P}$ be the pumping length given by the pumping lemma and (3) let $\boxed{S}$ be the $0^P 1^P$.

(4) with s being a member of C and having length more than p, the pumping lemma guarantees that $\boxed{s}$ can be split into three pieces $\rightarrow s = xyz$ where for any $i \geq 0$ the string xy^iz is in C.

we would like to show that this outcome is impossible Because $|xy| \leq p$ (by condition 3), $\boxed{y}$ must consist only of $\boxed{0's}$ so $xyyz \notin \boxed{C}$ $\rightarrow$ therefore, s cannot be pumped. $\rightarrow$ (ناقض) $\rightarrow$ فرض خلف $\rightarrow$ $\boxed{\text{نامنتظم است}}$

bis

چون تعداد صفرها بیش > تعداد یک‌ها بیش

★ نکته ی انتخاب رشته مهم است ★

مثلاً اگر برای مثال $\boxed{s}$ را $s = (01)^p$ در نظر گرفته بودیم به مشکل بر می‌خوردیم چون این رشته می‌تواند pump شود ما رشته‌ای می‌خواهیم که pump نشود مثلاً اگر $(x = \varepsilon)$ ، $(y = 01)$ و $(z = (01)^{p-1})$ $\leftarrow$

برای این رشته s $xy^iz \in C$ for every value of $\boxed{i}$

★ Using closure properties ★

An alternative way (method) of proving that C is nonregular follows from our knowledge that $\boxed{B}$ is nonregular. if $\boxed{C}$ were regular, $C \cap L(0^*1^*)$ also would be regular. $\rightarrow$ the reasons are that the language

اگر $M=(Q,\Sigma,\delta,q_0,F)$ یک DFA تشخیص زبان منظم A باشد ← می‌خواهیم ثابت کنیم لم پامپینگ برای A برقرار است.

p را تعداد استیت‌های M می‌گذاریم ← حالا باید نشان بدیم هر رشته‌ای مثل s که داخل A است و طولش حداقل برابر p است را می‌توانیم به سه بخش x, y, z شکست به طوری که این تجزیه ۳ شرط لم را ارضا کند.

سوال: اگر در زبان A هیچ کدام از رشته‌های به طول حداقل p نبود چی؟

اگر این اتفاق بیفتد ← صورت لم به طور بدیهی برقرار می‌شود (انتفای مقدم)

چون لم می‌گوید که:

اگر یک رشته‌ای باشد که طولش حداقل p باشد ← s را می‌توانیم تجزیه کنیم

p   q

اگر p کذب باشد ← انتفای مقدم می‌شود ← لم پامپینگ برای این زبان منظم برقرار است

هیچ رشته‌ای داخل زبان به طول بیشتر از p نیست

پس لم پامپینگ برای زبان‌های منظم برقرار است به دلیل انتفای مقدم

$\boxed{p = \text{تعداد استیت‌های (DFA) M}}$

فرض می‌کنیم s یک رشته داخل زبان A است که طولش حداقل p است

دنباله‌ی state هایی که هنگام خواندن رشته‌ی s داریم در نظر می‌گیریم ← مثلاً

$q_1, q_3, q_{20}, q_9, \ldots, q_{13}$ و چون s داخل زبان است

پس $\boxed{q_{13}}$ ← state پایانی است. ← $n \geq p$

فرض می‌کنیم اندازه‌ی $s = n$ ← $\boxed{|s| = n}$ و n حداقل برابر p است

حالا باید ~~طوری~~ نشون بدیم این S را می‌شه طوری تجزیه کرد که واحد اون ۳ شرط رو هست

پس stateهای که مشاهده می‌کنیم e

$q_1, q_3, q_{20}, q_9, \longrightarrow, q_{13}$

به طول n+1 الف

$S = \underbrace{s_1 \; s_2 \; s_3}_{y} \; s_4 \; s_5 \; \underbrace{s_6 \; \text{---} \; s_n}_{z}$

$q_1 \quad q_2 \quad q_{20} \quad q_9 \quad q_{17} \quad q_9 \quad q_6 \quad q_{35} \quad q_{13}$

طول دنباله‌ی این stateها $= n+1$

چون $n \geq p$ ← $\boxed{n+1 > p}$ ← چون توی DFA

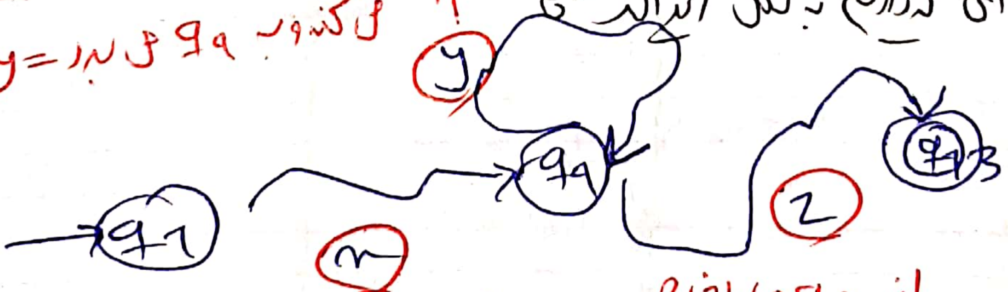
M تا p state بیشتر نداریم ← طبق اصل لانه‌ی کبوتری stateای هست که ما دوبار ملاقاتش کردیم ← مثلاً $\boxed{q_9}$

پس ادعا می‌کنیم که تجزیه‌ای که معرفی می‌کنیم ۳ شرط لم را ارضا می‌کند ←

$\boxed{q_9}$ e اون stateای هست که تکرار می‌شود

حالا تجزیه‌ای که داریم به شکل زیر است

تیکه‌ای که ما را از q_9 خارج می‌کند و به q_9 می‌برد $= y$



از q_9 ما را خارج می‌کند و به q_{13} می‌برد

تیکه‌ای از S که ما را از q_1 به q_9 می‌رساند

یعنی y را می‌توانیم اصلاً نداشته باشیم، می‌توانیم یک بار یا دوبار یا --- بار استفاده کنیم ازش و هر چند بار

که نیاز می‌توانیم ازش استفاده کنیم ← $\boxed{xy^2}$ داخل زبان الف هست $xyyz$

پس xy^2z داخل زبان نمی‌آمد ← شرط اول ارضا می‌شه .

شرط دوم ← y ناتهی است چون y قسمتی از k است که بین دوتا وقوع متفاوت از q_9 واقع شده ← حداقل یک کاراکتر هست که ما را از q_9 خارج می‌کند و دوباره ما را به q_9 برمی‌گرداند . ← اندازه‌ی y ناتهی

$\boxed{|y|>0}$

حالا شرط سوم برقرار است ؟

باید q_9 را اولین تکرار بگیریم ← طبق اصل لانه کبوتری در نخستین state $p+1$ از دنباله ← تکرار داریم ← وقتی $\boxed{xy}$ را می‌سازیم که اندازه‌اش کوچکتر مساوی p خواهد بود

$\boxed{|xy| \le p}$ می‌شود

برای اثبات منظم بودن ← DFA

← NFA

← عبارت منظم (regular expressions

برای اثبات نامنظم بودن یک زبان ← از لم پامپینگ استفاده می‌کنیم

از لم پامپینگ

★ نمی‌توانیم برای اثبات منظم بودن یک زبان استفاده کرد .

چون لم پامپینگ برای همه زبان‌های منظم برقرار است اما لزومی ندارد که هر زبانی که این لم برایش برقرار است منظم باشد (یعنی ممکن است یک زبان نامنظم باشد اما این لم هم برایش برقرار باشد) ←

از برقرار نبودن این لم ← نامنظم بودن را نتیجه می‌گیریم

عکس برقرار نیست

$L(0^*1^*)$ is regular and that the class of regular language is closed under intersection.
But $C \cap L(0^*1^*)$ equals B and we know that B is non-regular.

* گاهی اوقات کار بهتری راه برای اثبات نامنظم بودن یک زبان ای است که از closure theorems خواص بسته‌بودن استفاده کنیم. ← بسته‌بودن تحت اشتراک و complement

تکه اول و دوم مثل هم

example 3 $F = \{ww \mid w \in \{0,1\}^*\}$ → مثلا $\boxed{001001}$

(1) فرض می‌کنیم F منظم است ← (2) لم تزریق وعده می‌دهد که $\boxed{P}$ وجود دارد که رشته‌های زبان به طول حداقل P را می‌شود pump کرد

(3) $S = 0^P 1 0^P 1$ ← این رشته داخل زبان هست چون تکه اول و تکه دوم برابر هستند

برابر

(4) طول رشته S = $P < 2P+2$

→ pumping lemma طبق ← کد را تکه تکه می‌کنیم → $S = xyz$

ناقض است چون y شامل است و P تا کاراکتر اول رشته s همه‌شان صفر هستند و لذا لزوم می‌گوید که $\boxed{|xy| \le P}$ ← پس x و y هر دو از صفر تشکیل شده‌اند ← y فقط از صفر درست شده

$xyyz \notin F$ چون $xyz = 0^P 1 0^P 1$

اگر $y = 0^K$ و $K \ge 1$

$xy^2z = 0^{P+K} 1 0^P 1$ $\boxed{\notin F}$

xy می‌کنیم

به فرم ساده نیست

ناقض است

ww مخالف فرم

is a member of F

برای این مثال اگر s را $\boxed{0^p0^p}$ بگیریم حاصل pumping داخل زبان می‌افتد ولی رشته را می‌توان $\boxed{\text{پمپ}}$ کرد — برای اثبات نامنظم بودن فایده‌ای ندارد

در انتخاب رشته دقت کن

معکوس رشته — متقارن — ww^R — (0111)(1110)

example 1 $L = \{ww^R \mid w \in \{0,1\}^*\}$ is nonregular

Using the pumping lemma.

① let $\boxed{p}$ be the pumping length given by the pumping lemma. → طول ۲p+۲

② let $\boxed{s}$ be the string $\boxed{0^p110^p}$

چون $\boxed{s}$ عضو $\boxed{L}$ است $\boxed{\text{و}}$ (and) طول آن بزرگتر از $\boxed{p}$ است

لم تضمین می‌کند که s را می‌توان به سه قسمت تبدیل کرد

$s = xyz$ $\xrightarrow[\text{ارضا می‌کند}]{\text{که ۳ شرط را}}$ piece (y) must consist only of $\boxed{0's}$ (صفرها) →

$xyyz \notin L$ → ناقض ✗

$y = 0^k, k \ge 1$

طبق این شرط که $|y| > 0$

→ $xy^2z = 0^{p+k}110^p \notin L$

در انتخاب s دقت کن

نکته even though 0^p0^p is a member of $\boxed{L}$, it fails to demonstrate a contradiction because it can be pumped.

رشته‌های از 1 به طول مربع کامل

Example 5) Let $D = \{1^{n^2} \mid n \geq 0\}$.

$\boxed{D}$ contains all strings of 1's whose length is a perfect square

طول آنها

مربع کامل

از لم pumping استفاده می‌کنیم تا ثابت کنیم

فرض می‌کنیم طبق فرض خلف D منظم است $\Leftarrow$ ① p طول pumping باشد که توسط لم تزریق داده شده

② $s = 1^{p^2}$ $\rightarrow$ چرا چون (a) $\rightarrow$ یک عضو از D است

(b) $\rightarrow$ طول آن حداقل p است

③ پس طبق لم تزریق s می‌تواند به سه بخش تقسیم شود $\rightarrow$

$\boxed{s = xyz}$ $\longrightarrow$ for any $i \geq 0$ the string xy^iz is in $\boxed{D}$

④ $|s| = |xyz| = p^2$

$|xy^2z| = \underbrace{|xyz|}_{=p^2} + \underbrace{|y|}_{<p} \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$

① طبق شرط لم

$|xy| \leq p \rightarrow |y| \leq p$

② $|xy^2z| > p^2$

از 2 و 1 نتیجه می‌گیریم $\longrightarrow$ $p^2 < |xy^2z| < (p+1)^2$

طول xy^2z به صورت اکید بین دو مربع کامل قرار می‌گیرد

بین دو مربع کامل متوالی

خودش نمی‌تواند مربع کامل باشد

مثلاً $64 < n < 81$

n دیگر نمی‌تواند مربع کامل باشد

پس $\rightarrow$ $xy^2z \notin D$ $\rightarrow$ زبان D منظم نیست

رشته‌هایی که طول آنها توانی از ۲ است

Example 6.6 $A = \{a^{2^n} \mid n \geq 0\}$

زبان unary

① طبق فرض خلف، فرض می‌کنیم A منظم است ← p را طول pumping

در نظر می‌گیریم که توسط لم تزریق داده شده

② $s = a^{2^p}$ ← چون ① ← s عضو A و ② طول s

بیشتر از p

③ طبق لم تزریق ← s can be split into 3 parts

$s = xyz$ که ۳ شرط را ارضا می‌کند

④ شرط لم می‌گوید که $|xy| \leq p$ ← $|y| \leq p$ و $p < 2^p$ ←

$|y| \leq |xy| \leq p < 2^p$

$|xy^2z| = |xyz| + |y| < 2^p + 2^p = 2^{p+1}$

→ $|xy^2z| < 2^{p+1}$ و $|xy^2z| > 2^p$ → $2^p < |xy^2z| < 2^{p+1}$ →

طول رشته بین دو توان ۲ متوالی ← رشته xy^2z نمی‌تواند توانی از ۲

باشد → $\times$ تناقض ← نامنظم است

example 7 $L = \{w \in \{a,b\}^* \mid n_a(w) < n_b(w)\}$

is not regular → ((تعداد a ها اکیداً کمتر از تعداد b ها))

① pumping length ← p

② $s = a^p b^{p+1}$

③ $|xy| \leq p$ ← y فقط از $\boxed{a}$ تشکیل می‌شود

$y = a^k$ و $1 \leq k \leq p$ → اگر $i = 2$ → $a^{p+k} b^{p+1} \notin L$

$\times$ ← L منظم نیست.

تمرین ۶: با کمک لم تزریق نشان دهید کدام زبان زیر منظم نیستند؟

$L_1 = \{a^n b^m c^n \mid n \geq 0, m \geq 0\}$

$L_2 = \{a^n b^m : n \geq m\}$

Example 8 $B = \{0^m 1^n \mid m \neq n\}$ is not regular

روش اول: بهره‌گیری از خواص بستاری

فرض می‌کنیم که B منظم است ← طبق خواص بستاری، مکمل B ($\overline{B}$) هم regular است ← اشتراک با 0^*1^* هم باید منظم باشد

$\overline{B} \cap L(0^*1^*) = \{0^k 1^k \mid k \geq 0\}$

می‌دانیم که این زبان منظم نیست → B cannot be regular

روش دوم: بهره‌گیری از لم تزریق

$p =$ Pumping length ①

$p! = p(p-1)(p-2) \ldots 1$ $\quad s = 0^p 1^{p+p!} \in B$ ②

$|s| \geq p$

$x = 0^a \leftarrow s = xyz$ can be divided ③

④ $y = 0^b \rightarrow |y| > 0 \rightarrow (b \geq 1)$

$z = 0^c 1^{p+p!}$

$\rightarrow a+b+c = p$

اگر $i = \frac{p!}{b}$ و $s' = xy^{i+1}z$ ⑤

$y^i = 0^{p!} \rightarrow y^{i+1} = 0^{b+p!} \rightarrow s' = 0^{\underbrace{a+b+c}_{=p}+p!} 1^{p+p!}$

$(0^b)^{\frac{p!}{b}} = 0^{p!} \rightarrow s' = 0^{p+p!} 1^{p+p!} \notin B$

$m = n$ ← تناقض ← زبان نامنظم است

نکته مهم ۸ اگر یک زبان منظم باشد ← لم تزریق قطعاً برای آن برقرار است.
اما اگر لم تزریق برای یک زبان برقرار باشد ← آن زبان الزاماً منظم نیست
پس برای اثبات منظم بودن زبان نمی‌توان از لم تزریق استفاده کرد.

$\boxed{ex}$ $F = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}$

(a) show that F is not regular

(b) show that F acts like a regular language in the pumping lemma.

(c) explain why parts (a) and (b) do not contradict the pumping lemma.

نکته مهم ۹ شما حق تعیین m و x و y و z را ندارید چون با صور و جوابی مواجه نمی‌شوید.
مثلاً در اثبات نامنظم بودن زبان

$L = \{a^m b^n \mid \frac{m}{n} \text{ is an integer}\}$

بهره‌گیری از رشته‌ی $a^p b^p$ مناسب نیست
چون به ازای $y = a^p$ تناقض رخ نمی‌دهد (یعنی پاس کردن میسر است)
شما هم حق ندارید تعیین کنید که y چه هست یا نیست.
اما رشته‌ی $s = a^{p+1} b^{p+1}$ مناسب است چون به ازای هر تجزیه‌ای
دلخواه از s که واجد دو شرط $|xy| \leq p$ و $|y| \geq 1$ باشد داریم

$1 \leq |y| \leq p \longrightarrow xy^0z \notin L$

چون در رشته‌ی xy^0z تعداد a ها اکیداً کمتر از تعداد b هاست
و این تعداد برابر صفر هم نیست ← نسبت تعداد a ها به b ها
قطعاً صحیح نیست)

hirmandpaper

CFG

session 16

context-free grammars

the collection of languages associated with context-free grammars = context-free language (CFL)

زبان های مستقل از متن

زبان های منظم ، زیرمجموعه ی زبان های مستقل از متن هستند

rule

دارد یک زبان را توصیف می کند

مثال از یک گرامر مستقل از متن

$A \to 0A1$
$A \to B$
$B \to \#$

سه تا rule دارد

$\boxed{CFG}$

$n \in$ ... #را دارد ۲

هر گرامر ، متشکل از تعدادی rule است

substitution rule یا production

① هر rule در یک طرف ظاهر شده ... فقط متغیر در سمت چپ قرار دارد

② سمت چپ ، یک rule ، یک variable یا terminal است و سمت راست

متشکل از variable ها و other symbols

③ variable ها را با حروف بزرگ نشان می دهند

④ terminal ها را با حروف کوچک یا اعداد یا سمبل ها نشان می دهند

variable = A, B   terminal = 0, 1, #

ها (s) بالا

hirmandpaper

⑤ start variable = معمولاً سمت چپ اولین rule است

کار را از یک start variable آغاز می‌کنیم و کار با ساخت یک رشته ختم می‌شود.

به یک CFG (مثلاً مثال قبل) ← $\boxed{G_1}$ می‌گیم

string ها رو طبق یک گرامر (زبان) چطور ساخته می‌شوند ؟

① write down the start variable.

② find a var that is written down & a rule that starts with that var.
replace the written down var with the right hand side of that rule.

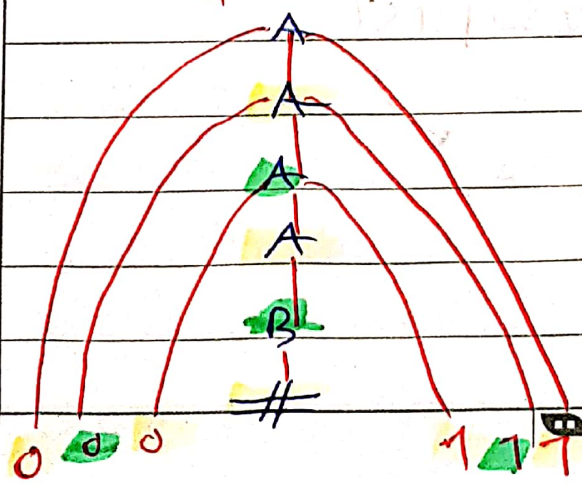
③ replace step ② until no var ~~remain~~ remain

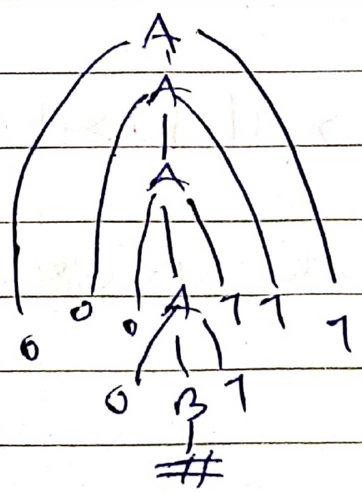
مثلاً G1

$$\underset{\text{start variable}}{A} \xRightarrow{\text{rule1}} 0A1 \xRightarrow{①} 00A11 \xRightarrow{①} 000A111 \xRightarrow{②} 000B111 \xRightarrow{③} 000\#111$$

به این کار derivation می‌گیم

$\boxed{\text{parse tree}}$





مجموعه های رشته هایی که تولید می گرامر دریک میشه ⟸ زبان تعیین این گرامر

$\boxed{L(G)}$ با نشان میدهیم

$L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$

برای راحتی ⟵ rule ها را گاهی اوقات ادغام میکنیم

$A \to 0A1$
$A \to B$

می توانیم بنویسیم

$A \to 0A1 \mid B$

or (|)

$\boxed{ex1}$ $S \to 0 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$

مثال از تولید رشته

$S \Rightarrow 0S1 \Rightarrow 01S11 \Rightarrow 010S011 \Rightarrow 0100011$

زبان تعیین گرامر

$L(G_1) = \{w \mid$ the length of w is odd and its middle symbol is a $0\}$

$\boxed{ex2}$ CFG ی پیدا کن که زبان زیر را تولید کند

$L = \{w \mid$ the length of w is odd$\}$

$\in \{0,1\}^*$

جواب:

$S \to 1 \mid 0 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$

دو

طول رشته هایی که ایجاد میکنه ⟵ $\boxed{2k+1}$

الف

$\boxed{ex3}$ $G = (\{S\}, \{a,b\}, R, S)$

$S \to aSa$

$S \to bSb$

$S \to \varepsilon$

derivation برای گرامر G:

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabbaa$

$L(G) = \{ww^R , w \in \{a,b\}^*\}$

زبان G مستقل الت

اما منظم نیست

نیست regular

$\boxed{ex4}$ یه CFG پیدا کن که زبان زیر را درک کند

$L = \{w \in \{0,1\}^* \mid w = w^R , w \text{ is a palindrome}\}$

$S \Rightarrow 1S1 \Rightarrow 101$ طول فرد

$S \Rightarrow 1S1 \Rightarrow 11$ طول زوج

جواب: $S \to \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

طول زوج لازم — طول فرد لازمه

$\boxed{ex5}$ یه CFG پیدا کن که زبان زیر را درک کند

L = {w | w starts and ends with same symbol}

رشته های هستند که با سمبل آغاز = پایان ، 0 و 1

جواب: $S \to 0 \mid 1 \mid 0T0 \mid 1T1$

$T \to 0T \mid 1T \mid \varepsilon$ هر رشته ای باشد دلخواهی را می‌تواند توصیف کند

$\boxed{00010}$ مثلاً برای تولید رشته را

$S \Rightarrow 0T0 \Rightarrow 00T0 \Rightarrow 000T0 \Rightarrow 0001T0 \Rightarrow$ $\boxed{00010}$ داریم

$\boxed{ex6}$ $L = \{0^n 1^{2n} \mid n \geq 0\}$

جواب: $S \to 0S11 \mid \varepsilon$

$$L(G) = \Sigma^* 1 \Sigma^* 1 \Sigma^* 1$$

$\boxed{ex7}$ زبان نقلی گرامر زیر چیه؟

$$S \to R1R1R1R$$
$$R \to 0R \mid 1R \mid \varepsilon$$

یه مثال از تولید رشته:

$$S \Rightarrow R1R1R1R \Rightarrow 1R1R1R \Rightarrow 11R1R \Rightarrow$$
$$110R1R \Rightarrow 1101R \Rightarrow 1101$$

پس می فهمیم زبان نقلی گرامر بالا:

$L(G) = \{w \mid w$ contains at least three 1s$\}$

حداقل ۳ تا ۱ دارد.

می تواند هر رشته باینری دلخواه را درست کند

$\boxed{\text{یه ترفند}}$ ← یک تکنیک داریم که اگر یک زبان پیچیده الف داریم برای

حل کردنش ← تکه تکه کنیم

$$S \to S_1 \mid S_2 \mid \ldots \mid S_k$$

که $\boxed{S_i}$ ها start variable گرامرهای

$\{0^n1^n \mid n \geq 0\}$ و $\{1^n0^n \mid n \geq 0\}$ خودشان هستند

مثال: اجتماع دو تا زبان

$$S_1 \to 0S_11 \mid \varepsilon$$
$$S_2 \to 1S_20 \mid \varepsilon$$

ادغام کن.

$$S \to S_1 \mid S_2$$
$$S_1 \to 0S_11 \mid \varepsilon$$
$$S_2 \to 1S_20 \mid \varepsilon$$

ex8 find a CFG that generates $L = \{a^n b^m \mid n \neq m\}$.

راه حل

حالت ① $n > m$

② $m > n$

تعداد a بیشتر

تعداد b بیشتر

$S \rightarrow AS_1 \mid S_1B$

$S_1 \rightarrow aS_1b \mid \varepsilon$

$A \rightarrow aA \mid a \longrightarrow a^k \quad k \geq 1$

$B \rightarrow bB \mid b \longrightarrow b^k \quad k \geq 1$

رشته‌هایی درست می‌کند که

رشته‌هایی درست می‌کند که تعداد مساوی a, b دارند

ex9 find a CFG for

$L = \{a^i b^j c^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$

X ← یک متغیری که همه‌ی رشته‌هایی که تعداد a و b یکسان دارند را تولید می‌کند

Y ← یک متغیری که ~ ~ ~ c, b ~ ~ تولید می‌کند

بخش اول ← تعداد a و b برابر اند

جواب ۲ تا را ادغام می‌کنیم ← حالا که

$S \rightarrow XC \mid AY$

بخش دوم ← تعداد b, c برابر

رشته‌هایی که a و b برابر دارند ← $X \rightarrow aXb \mid \varepsilon$

~ c, b ~ ~ ← $Y \rightarrow bYc \mid \varepsilon$

تولید a می‌کند ← $A \rightarrow Aa \mid \varepsilon$

فقط c تولید می‌کند ← $C \rightarrow Cc \mid \varepsilon$

ex 70) find a CFG that generates
$L = \{a^m b^n c^p d^q \mid m, n, p, q \geq 0$ and $m + n = p + q\}$

۳ حالات

تعداد مساوی a و d می سازد

$m = q \to n = p$ $\quad S \to aSd \mid A \mid B$

$m > q \to n < p$ $\quad A \to bAd \mid D$

$m < q \to n > p$ $\quad B \to aBc \mid D$

$D \to bDc \mid \varepsilon$

۳ تا حالت

تعداد مساوی b و c می سازد

① همچنین در تعریف rule از "$\rightarrow$" و در step های یک derivation از "$\Rightarrow$" استفاده می کنیم.

② $\Rightarrow^n$ ← یعنی صراحتاً n

$\alpha \Rightarrow^n \beta$

یعنی از α، n گام طی می کنیم می رسیم به β

③ $\Rightarrow^*$ یعنی طی صفر تا یک یا ... گام می رسیم به β مثلاً

if $G = (V, \Sigma, R, S)$ is a CFG, the $L(G) = \{x \in \Sigma^* \mid S \Rightarrow^*_G x\}$.

یک زبان G مستقل از متن است اگر یک گرامر مستقل از متن وجود داشته باشد که $L = L(G)$

گرامر مستقل از متن

Session 11

ex1 $A=\{a^n b^n \mid n\geq 0\}$ → مکمل = complement

مکمل A

$\overline{A} = \{a^m b^n \mid m \neq n\} \cup$ → اجتماع دو حالت

$L((a\cup b)^* b (a\cup b)^* a (a\cup b)^*)$

→ ۳ بخش

$S \to S_1 \mid S_2 \mid S_3$

$S_1 \to aS_1b \mid aS_1 \mid a$ → تعداد aها بیشتر

$S_2 \to aS_2b \mid S_2b \mid b$ → تعداد bها بیشتر

$S_3 \to XbXaX$

$X \to aX \mid bX \mid \varepsilon$ ← هر رشته ای با a و b تولید می کند

① $a^m b^n$ $m>n$ → S_1

② $a^m b^n$ $m<n$ → S_2

③ $(a\cup b)^* b (a\cup b)^* a (a\cup b)^*$ → S_3

X ← هر رشته‌ای با سایز دلخواه تولید می کند

ex2 find a CFG generating the following language:

$\{a^i b^j c^k \mid j \neq i+k\} \subseteq L(a^* b^* c^*) \subseteq \{a,b,c\}^*$

راه حل: $L_1 \cup L_2 = \{a^i b^j c^k \mid j > i+k\} \cup$
$\{a^i b^j c^k \mid j < i+k\}$

$\boxed{S \to S_1 \mid S_2}$

باهر a دو تا b داریم که   b اضافه می کنه

$L_1 = MNP = \{a^i b^i \mid i \geq 0\} \circ \{b^n \mid n \geq 0\} \circ \{b^k c^k \mid k \geq 0\}$ concate

b و c مساوی تولید شوند   a و b مساوی تولید کنن

$S_1 \rightarrow S_M S_N S_P$, $S_M \rightarrow a S_M b \mid \varepsilon$

فقط b تولید می کنه $S_N \rightarrow b S_N \mid b$

$S_P \rightarrow b S_P c \mid \varepsilon$

ترتیب مهم میشه

b و c مساوی تولید می کنن

$S_1 \rightarrow S_M S_N S_P$ با

$S_1 \rightarrow S_M \mid S_N \mid S_P$ فرق داره

$L_2 = L_3 \cup L_4 = \{a^i b^j c^k \mid j < i\} \cup \{a^i b^j c^k \mid i \leq j < i+k\}$

$S_2 \rightarrow S_3 \mid S_4$

$L_3 = QRT = \{a^i \mid i > 0\} \circ \{a^i b^i \mid i \geq 0\} \circ \{c^i \mid i \geq 0\}$

$S_3 \rightarrow S_Q S_R S_T$, $S_Q \rightarrow a S_Q \mid a$

$S_R \rightarrow a S_R b \mid \varepsilon$ $S_T \rightarrow c S_T \mid \varepsilon$

$L_4 = UVW = \{a^i b^i \mid i \geq 0\} \circ \{b^i c^i \mid i \geq 0\} \circ \{c^i \mid i > 0\}$

$S_4 \rightarrow S_U S_V S_W$ $S_U \rightarrow a S_U b \mid \varepsilon$

$S_V \rightarrow b S_V c \mid \varepsilon$

$S_W \rightarrow c S_W \mid c$

ex3) $\{ w\#x \mid w^R \text{ is a substring of } x$

for $w, x \in \{0,1\}^*\}$.

Solution:

قسمت جدا، reverse را میسازد

$S \to TX$

w^R هم w را میسازد هم ← $T \to 0T0 \mid 1T1 \mid \#X$

$X \to 0X \mid 1X \mid \varepsilon$

001 # 111 100 0010

w w^R

هر رشته‌ای دلخواه را تولید میکند

رشته‌ای بس # و w^R

110 # 000 011 0101

w w^R

$\boxed{w, w^R}$

ex4) create a CFG to generate a non_palindrome over $\{a,b\}$

$\boxed{w\#w^R}$ a و b match نشوند

راه 1) $S \to aSa \mid bSb \mid aTb \mid bTa$

طول زوج

$T \to aTa \mid aTb \mid bTa \mid bTb \mid \varepsilon \mid a \mid b$

هر اتفاقی بیفتد مهم نیست

طول فرد

راه 2) ولی رشته a ولی رشته b

$R \to XRX \mid S$

$S \to aTb \mid bTa$

$T \to XTX \mid X \mid \varepsilon$

$X \to a \mid b$

معرفی عدم تقارن

$R \Rightarrow XRX \Rightarrow XXRXX \Rightarrow XXSXX \Rightarrow XXaTbXX$

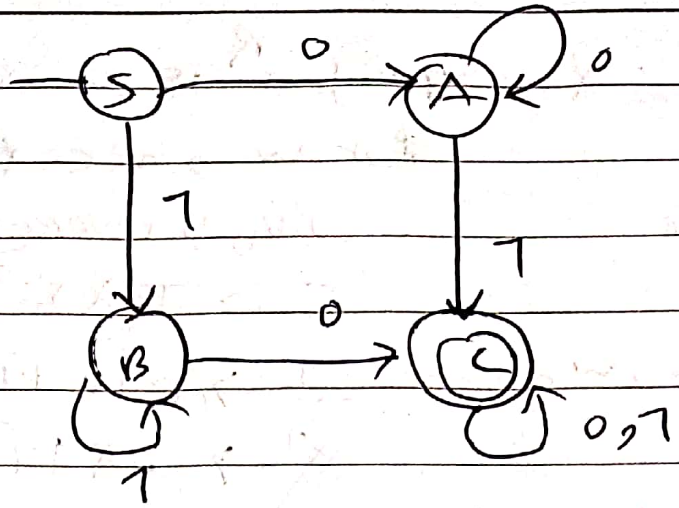
$\Rightarrow XXaXTXbXX \to XXaXXbXX$

عدم تقارن

hirmandpaper

Scanned by CamScanner

ex)

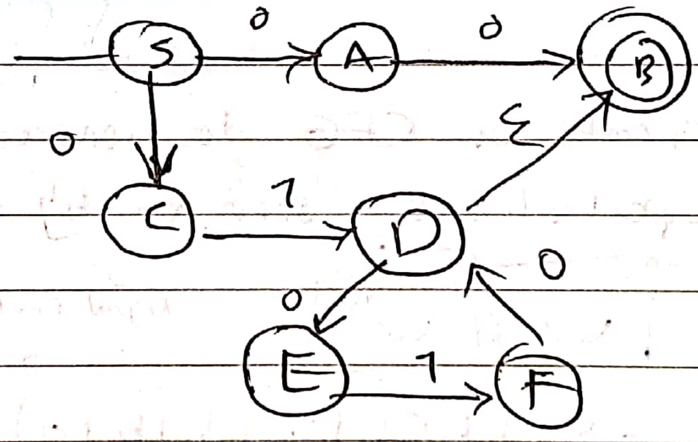


$S \to 0A \mid 1B \qquad A \to 0A \mid 1C$

$B \to 1B \mid 0C \qquad C \to 0C \mid 1C \mid \varepsilon$

چون state ها فاینال اند ← با ε ختم شد

ex)



$S \to 0A \mid 0C \qquad A \to 0B \qquad B \to \varepsilon$

$C \to 1D \qquad D \to 0E \mid B \qquad E \to 1F$

$F \to 0D$

خواص بستاری زبان های مستقل از متن

CFL ها تحت عملگرهای اجتماع و concat و star بسته است

اما ← برای عملگرهای مکمل و اشتراک بسته نیست

$L_1 = \{a^n b^n c^m \mid n \geq 0 , m \geq 0\}$

$S \rightarrow S_1 S_2$

$S_1 \rightarrow a S_1 b \mid \varepsilon$

$S_2 \rightarrow c S_2 \mid \varepsilon$

$L_2 = \{a^n b^m c^m \mid n \geq 0 , m \geq 0\}$

$S \rightarrow S_1 S_2$

$S_1 \rightarrow a S_1 \mid \varepsilon$

$S_2 \rightarrow b S_2 c \mid \varepsilon$

معروف ترین زبان

non_CFL

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ → اشتراک بسته نیست

$\boxed{\text{نیست}}$ context_free

مکمل هم لزوماً نیست CFL

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

با نقض خلف می رسیم

یک زبان non_CFL معروف دیگر

$L_1 = \{ww \mid w \in \{a,b\}^*\}$

ww^R CFL هست اما ww نیست

راست‌ترین

Leftmost & Rightmost Derivation

چپ‌ترین

session 12

in the grammar G

$$S \to a \mid S+S \mid S*S \mid (S)$$

③ ② ①

همه‌ی عبارات جبری که دربردارنده‌ی a و عمل ضرب و جمع و پرانتز هستند

مثال: ① $a+(a*a)$

یک leftmost derivation برای $a+a*a$ ← بنویسید derivation

الف که ما هر بار چپ‌ترین var را برای جایگزینی انتخاب می‌کنیم

$$S \Rightarrow S+S \Rightarrow a+S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a$$

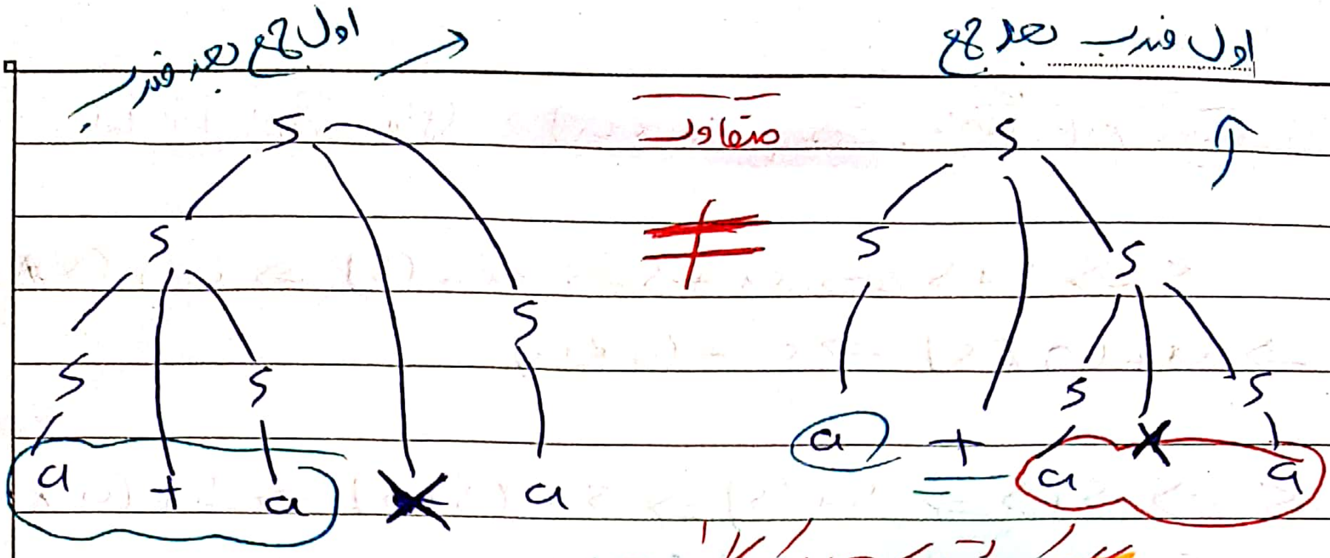
$$S \Rightarrow S*S \Rightarrow S+S*S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a$$

ابهام ← خوب نیست ریدرایو از یک رشته چند تا راه داریم

مثال: $S \to S+S \mid S\times S \mid (S) \mid a$

این گرامر رشته‌ی $a+a\times a$ را دو طور می‌سازد مبهم است

دو تا parse tree مختلف دارد



اگر یک رشته که تولید می‌کند گرامر تولید،
مثلاً، دو تا Parse tree داشته باشد ← مبهم است

تولید یک گرامر که نامبهم است:

$S \to S+T \mid T$

$T \to T \times F \mid F$

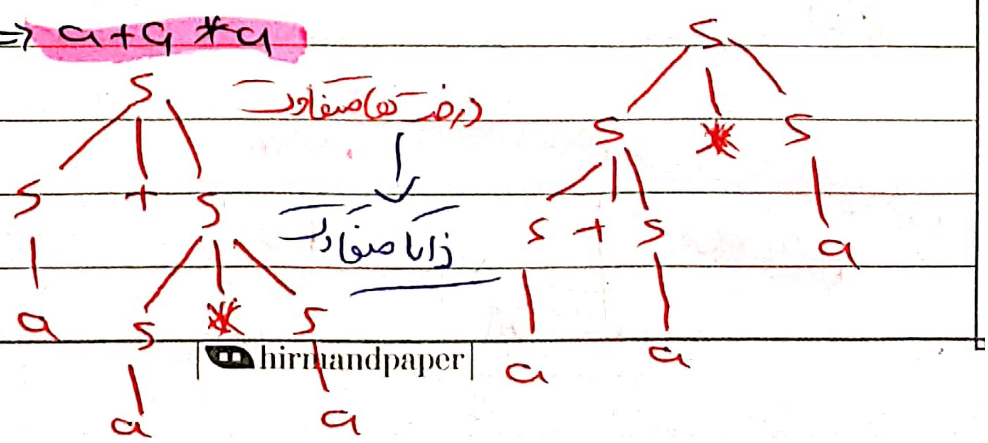
$F \to (S) \mid a$

نکته: درایویشن‌هایی که درخت تجزیه آن‌ها یکسان است، ماهیتاً تمایز عمده‌ای باهم ندارند و صرفاً ترتیب به کارگیری اول‌ها در آن‌ها متفاوت است.

اما درایویشن‌هایی که درخت تجزیه متفاوت دارند ← ماهیتاً متفاوت‌اند.

مثال: drivation1: $S+S \Rightarrow a+S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a$

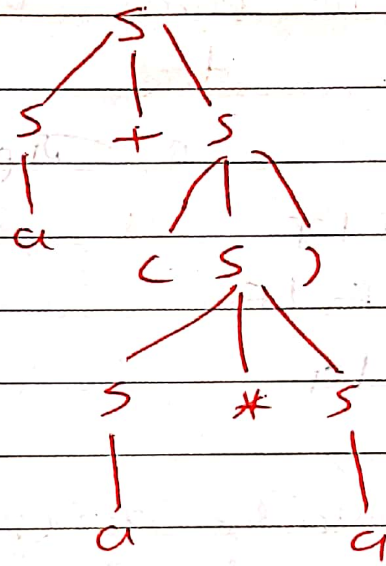
drivation2: $S \Rightarrow S*S \Rightarrow S+S*S \Rightarrow a+S*S \Rightarrow a+a*S \Rightarrow a+a*a$



اما ۲ تا درایوشن برای $a+(a\times a)$ که ماهیتاً متمایز نیستند

$S \Rightarrow S+S \Rightarrow a+S \Rightarrow a+(S) \Rightarrow a+(S*S)$
$\Rightarrow a+(a*S) \Rightarrow a+(a*a)$

$S \Rightarrow S+S \Rightarrow S+(S) \Rightarrow S+(S*S) \Rightarrow S+(a*S)$
$\Rightarrow S+(a*a) \Rightarrow a+(a*a)$



مفهوم ابهام LMD و RMD

if G is a CFG then for every $x \in L(G)$ these 3 statements are equivalent

1. x has more than one derivation tree
2. " " " than " leftmost derivation
3. " " " " rightmost "

اگر ۲ تا بیش تر LMD یا RMD داشته باشیم

گرامر مبهم است

ابهام یک صفت برای گرامر — ذاتاً مبهم برای زبان

اگر هر گرامری که برای یک زبان می‌سازیم مبهم باشد آنگاه زبان ذاتاً مبهم است.
همه‌ی گرامرها

الف) $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$ → ذاتاً مبهم

فرم چامسکی

★ هر کدام CFG را می‌توان به CNF برد.

(تعریف فرم نرمال چامسکی)

if every rule is of the forme

$S \to \varepsilon$ یا
$A \to BC$
$A \to a$

var ها غیر از start var

سمبل

برای rule ها باید سمت راست ۲ تا var بجز خود الفبا با یک سمبل

$\boxed{A \to aX}$ جایز نیست

ابتدا rule هایی که به این ۲ فرم نیستند را حذف می‌کنیم و اثر حذف را جبران می‌کنیم

① اضافه کردن یک الفبا جدید ② همه‌ی rule های به فرم $A \to \varepsilon$ به فرم ε_rules حذف می‌کنیم

③ rule های که سمت راست یک var واحد وجود دارد را حذف می‌کنیم (مثل $\boxed{A \to B}$)

حذف renaming rule

hirmandpaper

④ rule هایی که طولشان بزرگ است حذف می‌شوند

اولین گام : $S_0 \to S$ به عنوان start var

دومین گام : $\boxed{A \to \varepsilon}$ ها را حذف می‌کنیم جایی که A ایستاد

تست

سپس برای هر A که سمت راست است ما یک rule جدید با occurrence ای که پاک کردیم و تکرار می‌دهیم.

یعنی $A \to \varepsilon$ را پاک کردیم

$R \to uAv \mid uv$

اضافه می‌کنیم

$R \to uAvAw \xrightarrow[A \to \varepsilon]{\text{برای جبران اثر حذف}}$ $R \to uvAw$ / $R \to uAvw$ / $R \to uvw$

همه را اضافه می‌کنه

گام سوم : حذف unit rule ها

$A \to B$ و $B \to \Delta \mid \square \mid 0$

را می‌خواهیم حذف کنیم

اثر حذف می‌کنه $\to$ $A \to \Delta \mid \square \mid 0$

اگر یک بار یک rule را حذف کردیم و دوباره ظاهر شد نیازی به حذف دوباره نیست

گام ۴ : حذف rule های long

$X \to aAbB$

session 13

Removing useless productions

مثلاً در گرامر $S \to aSb \mid \varepsilon \mid A$, $A \to aA$

$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow$ ... نمیرود

افتادن از S به A ،
$S \to A$
هیچ نتیجه ای ندارد

یعنی با رفتن از S به A ، هیچ وقت به یک رشته (sentence) نمی رسیم ← هیچ نقشی ایفا نمی کند

با حذف $S \to A$ ↓ :
$S \to aSb \mid \varepsilon$

حذف rule های بی فایده یا فایده دار است؟

تعریف ← اگر $G = (V, \Sigma, R, S)$ یک context free grammer باشد

متغیر A که $A \in V$ مفید است اگر و تنها اگر حداقل یک w که $w \in L(G)$ وجود داشته باشد که

$$S \Rightarrow^* xAy \Rightarrow^* w$$

اگر حداقل یک رشته باشد که A در آن ظاهر شود ← به متغیر A سودمند گویند

با x, y in $(V \cup \Sigma)^*$

a variable is useful if & only if it occurs in at least one derivation

یک rule که یک متغیر (تولید کننده) useless باشد ← useless است

هدف حذف این rule های بی فایده

۲ عامل هست که باعث میشه ماشین یه متغیر و الودمند الت

① generating باشد — مولد و یا زنده باشد

یعنی متغیر A، بتواند خودش را به رشته‌ای از terminal ها برساند

A is generating if $A \Rightarrow^* w$ for some term string w.

② reachable باشد

متغیر A زمانی قابل دسترسی است که یک derivation به صورت

$S \Rightarrow^* \alpha X \beta$ for some $\alpha \in (V \cup \Sigma)$ و
$\beta \in (V \cup \Sigma)^*$

متغیر A، باید طوری باشد که در دسترس start باشد

پس اول non generating ها را حذف می‌کنیم
سپس unreachable ها

ex) $S \to A \quad A \to aA | \varepsilon \quad B \to bA$

$B \to bA \Rightarrow b \in \Sigma^*$ generating ← B , A

reachable ← A اما B نیست

از S نمی‌توانیم به B برسیم

generating generating

ex) $S \to aS | A | C \,,\ A \to a \,,\ B \to aa \,,\ C \to aCb$

non_generating C ← C را و قواعدی که دارای rule هست C دارای

$S \to aS | A$

هستند را حذف می‌کنیم پس

$A \to a$

② از S به B نمی‌توانیم برسیم

$B \to aa$

non_reachable ← B

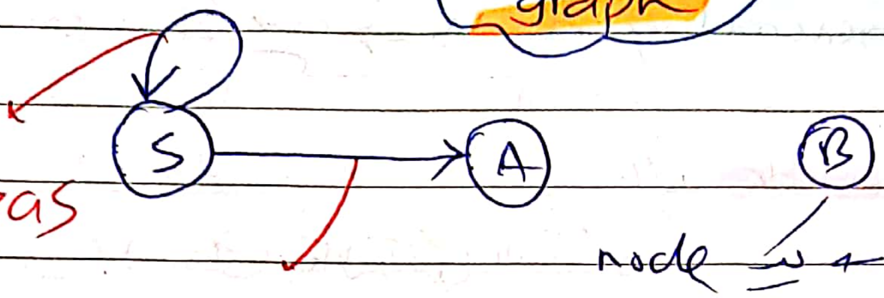
B را حذف می‌کنیم

B → جوابش تهی

$S \to aS \mid A$
$A \to a$

برای مشخص قابل دسترسی بودن یک متغیر → می‌توانیم گراف رسم کنیم

dependancy graph — گراف وابستگی



$S \Rightarrow aS$

برای B → حذف می‌کنیم

تصویر هر متغیر → یک node

$S \Rightarrow A$

پس توی گراف → اگر و تنها اگر از راس C به D یال داریم که

$C \to aD\gamma$

راس‌هایی که از S به آن‌ها یک مسیر وجود دارد قابل دسترسی هستند

راس‌های

در گام اول non generating ها حذف می‌شوند در گام دوم → unreachable ها

در گام اول → اون متغیرهایی اضافه می‌شن که به terminal ها ختم می‌شن (ε یا رشته)

اول اونایی که کاملا ترمینال هستند

راس‌های که ترکیب حالتی هستند → مثل اینها

$A \to \varepsilon$ ✓ مولد
$B \to ba$ ✓ مولد
$C \to abb$ ✓ مولد
$D \to abac$ ✓ مولد
$X \to AaaBD$ ✓ مولد
$Y \to AXaaC$ ✓ مولد

در گام دوم → اونایی که مولد نیستند را حذف می‌کنیم

سپس نوبت حذف unreachable ها میرسد

$S \to AaaB$ (A و B در دسترس)

$A \to XaaY$ (X و Y هم در دسترس)

ex:

✓ $S \to AC \mid BS \mid B$

✓ $A \to aA \mid aF$ → $A \to aF$

✓ $B \to \cancel{CF} \mid b$

✗ $\cancel{C \to cC \mid D}$

✗ $\cancel{D \to aD \mid BD \mid C}$

✓ $E \to aA \mid BSA$ A و S و B هم generating

✓ $F \to bB \mid b$

① حذف های متغیرهای غیرمولد

E هم generating است

C و D متغیر غیرمولد ← باید rule های شامل C و D را هم حذف کنیم ←

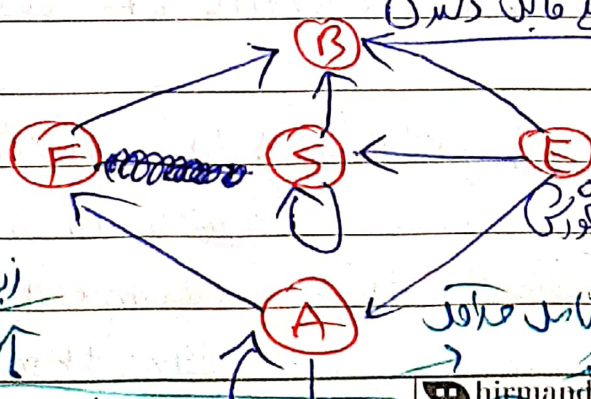
$V_1 = \{S, A, B, E, F\}$

$\Sigma_1 = \{a, b\}$

$R_1 = \{S \to BS \mid B,\ A \to aA \mid aF,\ B \to b,$
$E \to aA \mid BSA,\ F \to bB \mid b\}$

گام دوم ← حذف های متغیرهای غیرقابل دسترس

به کمک گراف وابستگی



طبق این گراف ← S فقط به خودش و B می‌رود

قابل حذف

زبان کدام

$L(\hat{G}) = L(b^+)$

b

$S \to SB \mid B$
$B \to b$

نکات مهم: حتما باید اول غیرسازنده ها را حذف کنیم و بعد غیرقابل دسترسی ها را

unreachable — non_generating

(مسائل CFL که undecidable هستند)

الگوریتمی نیست برای حل آنها → مسائلی که نمی‌توانیم الگوریتمی حل کنیم

$(AMBIG_{CFG})$

① is a given CFG (G) ambiguous?

مسئله الگوریتمی که یک context free gramer را بگیرد و تعیین کند که مبهم هست یا نه G وجود ندارد ← undecidable است

② Is a given CFL inherently ambiguous?

زبان تعریف شده CFL ذاتا مبهم هست یا خیر

③ Is the intersection of two CFL's empty?

الگوریتمی نداریم که G_1 و G_2 را بگیرد و بگوید اشتراکی بین G_1 و G_2 وجود دارد یا خیر

④ Are two CFL's the same? (EQ_{CFG})

G_1 و G_2 را بگیرد → آیا $L(G_1) = L(G_2)$?

⑤ Is a given CFL equal to Σ^*, where Σ is the alphabet of this language?

(ALL_{CFG})

نمیشه الگوریتم برای اون طراحی کرد

Part 3

decidable یعنی از مسائل

Some decidable properties of context free languages.

کدام‌ها وارد دارد

$\boxed{A_{CFG}}$ — مسئله‌ای است که آیا $\boxed{w}$ داخل $\boxed{L(G)}$ وجود دارد یا خیر.

acceptance — مسئله‌ی عضویت (membership algorithm)

یا نه کدام را قرار می‌گیرد

$\boxed{CYK}$ مهم‌ترین الگوریتم عضویت است. می‌توانیم تعیین کنیم آیا w داخل G در زبان هست

emptiness

E CFG ؟

آیا زبان نظیر یک گرامر داده شده ، تهی هست یا خیر ← empty or not ؟

decidable است

INFINITE یک CFG را به عنوان ورودی به ما می‌دهند ؟ و از ما می‌خواهند که تست کنیم آیا زبان نظیر این گرامر (CFG) بی‌نهایت است یا متناهی است

infinite or not ؟

brute force حل برای تست کردن عضویت (membership) به شکل ناکارآمد است ؟ و الگوریتم که ایجاد می‌شود از order بسیار بالایی است

در این نوع ؟ سعی می‌شود از همه‌ی حالات ممکن استفاده کند و همه را در نظر بگیرد و وقتی همه‌ی حالات را ساخت ؟ تصمیم گیری را انجام دهد ← در نظر گرفتن همه‌ی حالات ممکن خیلی زمان می‌برد ← خوب نیست

مثال ؟ یک رشته به نام $\boxed{w}$ که طول آن $\boxed{n}$ است و یک گرامر داریم که در فرم چامسکی است

$G = (V, \Sigma, R, S)$

CNF

آیا $\boxed{S}$ ، $\boxed{w}$ را می‌سازد ؟

برای ساختن رشته‌ای به طول $\boxed{n}$ ← rule های با فرم $\boxed{A \to BC}$ را $\boxed{n-1}$ بار فراخوانی کنیم

و سپس rule های که به فرم $\boxed{A \to a}$ هستند را $\boxed{n}$ بار فراخوانی کنیم

در مجموع ؟

$n-1+n = \boxed{2n-1}$ تا rule باید استفاده کنیم

برای عمل کردن به شکل brute force میگیم یک رشته‌های دارم که $\boxed{2n-1}$ تا rule دارد که باید به کار گرفته شوند تا که w ها را بسازند

w به شکل $r_1 r_2 \ldots r_{2n-1}$

پس همه‌ی رشته‌های به طول $\boxed{2n-1}$ را می‌سازم بعد یکی یکی که داشتیم رشته‌ها را باید چک کنیم هرکدام می‌تواند $\boxed{w}$ را بسازد یا نه.

اگر گرامر ما دارای $\boxed{m}$ تا rule باشد ($|R| = m$)

تعداد کل رشته‌هایی که باید درنظر بگیریم تا بفهمیم w را می‌سازد یا نه

m^{2n-1}

$2n-1$ حالت داریم که برای هر حالت هم $\boxed{m}$ تا حالت داریم

نمائی است → کارا نیست

راه حل w الگوریتم CYK → dynamic programming است

از $\boxed{O(n^3)}$ است

طول رشته = n

مثل‌های الگوریتم‌های dynamic programming یک جدول داریم که باید خانه‌هایش را پر کنیم

	طبقه 1	رشته‌ی هدف			
سطر پنجم	X_{15}	طبقه 2		X_{ij} ها → هرکدام یک	
سطر چهارم	X_{14}	X_{25}	طبقه 3	مجموعه از متغیرها هستند	
سطر سوم	X_{13}	X_{24}	X_{35}	طبقه 4	$\boxed{X_{ij} \subseteq V}$
سطر دوم	X_{12}	X_{23}	X_{34}	X_{45}	طبقه 5
سطر اول	X_{11}	X_{22}	X_{33}	X_{44}	X_{55}

w = a_1 a_2 a_3 a_4 a_5

طول رشته = 5

شماره ... ← $X(i,j)$ ← لتیق

اگر X_{ij} دارای یک متغیری باشد ← یعنی اون متغیر می‌تواند زیررشته‌ی $\boxed{a_{i+1} \text{ تا } a_j}$ از w را تولید کند

یک ← Sub_string

X_{ij} is the set of variables $\boxed{A}$ such that
$A \Rightarrow^{*} a_i a_{i+1} \dots a_j$

$\boxed{X_{15}}$ ← دارای متغیرهای الف که قادر هستند که $\boxed{a_5 \text{ تا } a_1}$ را بسازند ← یعنی X_{15} کل رشته‌ی w را می‌تواند بسازد ← وقتی جدول را کامل کردیم باید بشیم که $\boxed{S}$ داخل متغیرهای X_{15} هست یا خیر

پس وقتی $\boxed{X_{1n}}$ ← باید جدول ← چک می‌کنیم آیا $S \Rightarrow^{*} w$ یا S که X_{1n} هست یا خیر

اگر باشد یعنی S می‌تواند a_1 تا a_n (کل w) را بسازد

واگر نباشد ← نمی‌تواند w را بسازد

جدول ها لغزه‌به‌بعد از پایین به بالا پر می‌شود

تعداد درایه‌ها $\frac{n(n+1)}{2}$

کاری که صورت می‌گیرد برای پیدا کردن هر کدام از درایه‌های جدول

$\boxed{O(n)}$

$O(n) \times O(n^2) = \boxed{O(n^3)}$

نام مهم الگوریتم $\boxed{CYK}$ ← فرم گاصر وارد ← حامسی

برای اینکه بفهمیم چه متغیرهای توی X_{ij} قرار می‌گیرند چه هست

اگر $A \Rightarrow^* a_i a_{i+1} \ldots a_j$ قدرت [illegible] از a_i تا a_j را بسازد

B باید طوری [illegible]

جا مسیر

$A \to BC$ پس

B باید بخش اول رشته را بسازد

C ـــ ـــ دوم ـــ ـــ

پس برای اینکه تصمیم بگیریم آیا A قادر هست که زیر رشته‌ی a_i تا a_j را بسازد یا نه باید نشان کنیم که آیا rule ای به فرم $\boxed{A \to BC}$ در مجموعه rule های ما هست یا خیر که B بتواند تکه اول و C بتواند تکه دوم را بسازد

اگر $\boxed{i \le k < j}$

و $B \in X_{ik}$ قبلاً محاسبه

$\boxed{\text{از } i \text{ تا } k}$

$C \in X_{k+1,j}$

$\boxed{\text{از } k+1 \text{ تا } j}$

و ③ $A \to BC \in R$

X_{ik} و $X_{k+1,j}$

قبلاً محاسبه کرده

مثال ها $a_1 \; a_2 \; a_3 \; a_4$

برای $\boxed{X_{14}}$ ← باید ببینیم آیا متغیری هست که B تکه اول و C تکه دوم را بسازد ←

$\underbrace{a_1}_{B} \; \underbrace{a_2 \; a_3 \; a_4}_{C}$

$A \to BC$

$\underbrace{a_1 \; a_2}_{B} \; \underbrace{a_3 \; a_4}_{C}$

$\underbrace{a_1 \; a_2 \; a_3}_{B} \; \underbrace{a_4}_{C}$

اگر همچین چیزی وجود داشت

متغیر $\boxed{A}$ را می‌گذاریم داخل $\boxed{X_{14}}$

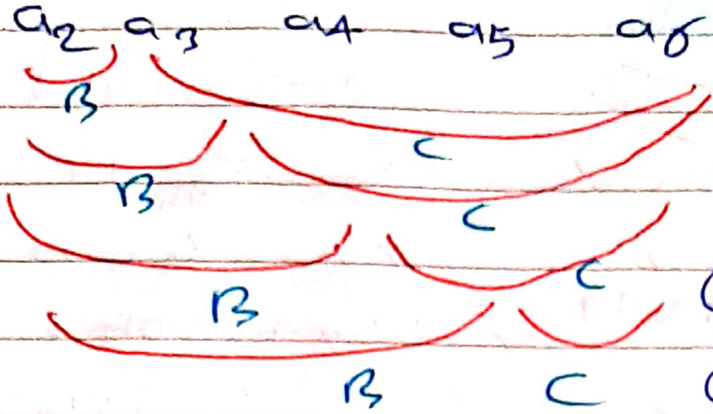
$\boxed{X_{ik}}$ (در ردیف)

$k - i + 1$ در ردیف

قرار دارد

hirmandpaper

مثلاً برای محاسبه‌ی $\boxed{X_{26}}$ ←

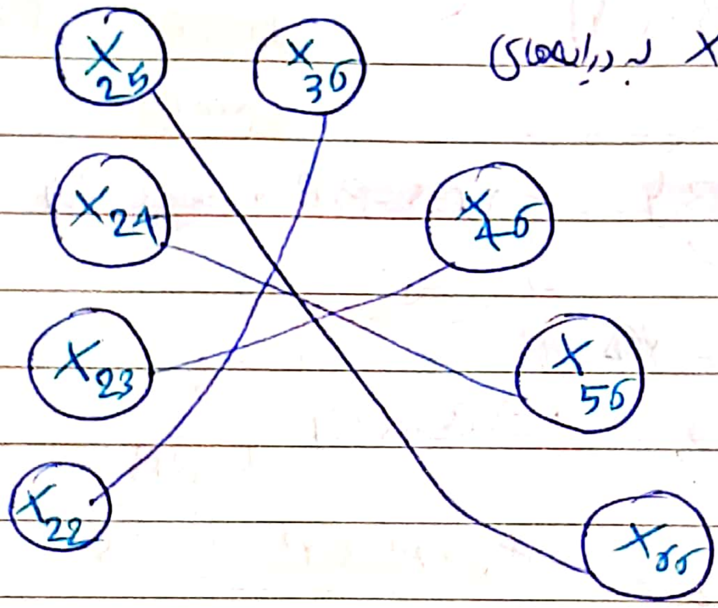


اولی → (X_{22}, X_{36})

دومی → (X_{23}, X_{46})

سومی → (X_{24}, X_{56})

چهارمی → (X_{25}, X_{66})



پس برای محاسبه‌ی X_{26} به درایه‌های زیر نیاز داریم که قبلاً حساب شدند

$\boxed{ex}$

$S \to AB \mid AC \mid AA$,

$A \to CB \mid a$,

$B \to AC \mid b$,

$C \to CC \mid b$.

از بین اینها کدام متغیرها $\boxed{b}$ را می‌سازند ؟

$\downarrow$

$\boxed{B, C}$

A ← a را می‌سازد و کدام‌ها

5	{S, A, B}				
4	{S, A}	{S, A, B}			
3	{S}	{A}	{S, B}		
2	{A, C}	∅	{S, B}	{A, C}	
1	{B, C}	{B, C}	{A}	{B, C}	{B, C}
	X_{11}	X_{22}	X_{33}		
strings	1	2	3	4	5
	b	b	a	b	b

برای محاسبه‌ی X_{25} ؟

$a_2 \quad a_3 \quad a_4 \quad a_5$

باید به درایه‌های

(X_{22}, X_{35}) ،

(X_{23}, X_{45}) ،

(X_{24}, X_{55})

دسترسی داشته باشیم

$(\{B, C\}, \{S, B\})$

$(\varnothing, \{A, C\})$

$(\{A\}, \{B, C\})$

بخش اول prefix — بخش دوم suffix

$\{B, C\} . \{S, B\} = \{BS, BB, CS, CB\}$

در کنار هم قرار دادیم

$\varnothing . \{A, C\} = \varnothing$

آیا متغیری هست که من را

$\{A\} . \{B, C\} = \{AB, AC\}$

BS یا BB یا CS یا CB سراغ

به rule های گرامر نگاه کن

$\cancel{BS}, \cancel{BB}, \cancel{CS}, \boxed{CB}$ ✓

توی rule ها $\boxed{CB}$ داریم ←

CB → B C ← X_{22} را می‌سازد

B B ← X_{35} را می‌سازد

$\boxed{A}$ را به X_{25} اضافه می‌کنیم

آیا rule ای داریم که سمت راستش AB یا AC باشد؟

بله ← $\boxed{S}$ می‌گذارد ← $\boxed{S}$ را به X_{25} اضافه می‌کنیم

$S \to AB$

$S \to AC$

$B \to AC$

S و B را هم اضافه می‌کنیم

جواب

$X_{25} = \{S, A, B\}$

ex6

$S \to AB \mid BC$ — we shall test for

$A \to BA \mid a$ — membership in $\boxed{L(G)}$

$B \to CC \mid b$ — the string

$C \to AB \mid a$ — $\boxed{baaba}$

	{S, A, C}				
X_{14}	∅	{S, A, C}			
X_{13}	∅	{B}	{B} → X_{34}		
	{S, A}	{B}	{S, C}	{S, A}	
	{B}	{A, C}	{A, C}	{B}	{A, C}
	b	a	a	b	a

X_{11}, X_{22}, X_{33}

$A \to a$, $C \to a$, $\boxed{B \to b}$

برای تولید a ← {A, C}

b ← {B}

برای محاسبه $\boxed{X_{24}}$

$a_2 \; a_3 \; a_4$

(X_{22}, X_{34}) یا

(X_{23}, X_{44})

$(X_{22}, X_{34}) \Rightarrow \{A, C\}.\{S, C\} = \{\cancel{AS}, \cancel{AC}, \cancel{CS}, CC\}$

$(X_{23}, X_{44}) \Rightarrow \{B\}.\{B\} = \boxed{\{BB\}}$

حالا توی rule های گرامر ببینیم کدام یک از ترکیب های بالا هست

فقط CC وجود دارد ← متغیر $\boxed{B}$ را به $\boxed{X_{24}}$ اضافه می کنیم

تمرین B متغیرهایی که اضافه‌اند (حذف متغیرهای useless)

G: $S \to aA \mid \cancel{BD}$

✓ $A \to aA \mid \cancel{aAB} \mid aD$

✗ ~~$B \to aB \mid aC \mid BF$~~

✗ ~~$C \to Bb \mid aAC \mid E$~~

✓ $D \to bD \mid \cancel{bC} \mid b$

✗ ~~$E \to aB \mid bC$~~

✓ $F \to aF \mid aG \mid a$

✓ $G \to a \mid b$

$S \to aA \mid \cancel{BD}$

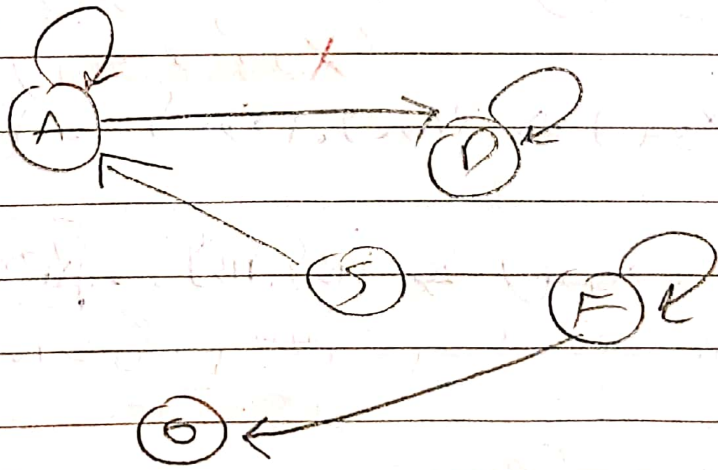
$A \to aA \mid aD$

$D \to bD \mid b$

$F \to aF \mid aG \mid a$

$G \to a \mid b$

حالا نوبت حذف unreachable ها رسید



پس تنها $\boxed{A}$ و $\boxed{D}$ قابل دسترسی اند ←

$S \to aA$

$A \to aA \mid aD$ → جواب

$D \to bD \mid b$

session 14

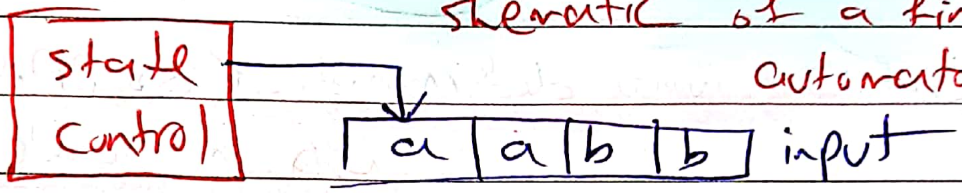
PDA

Push down automaton

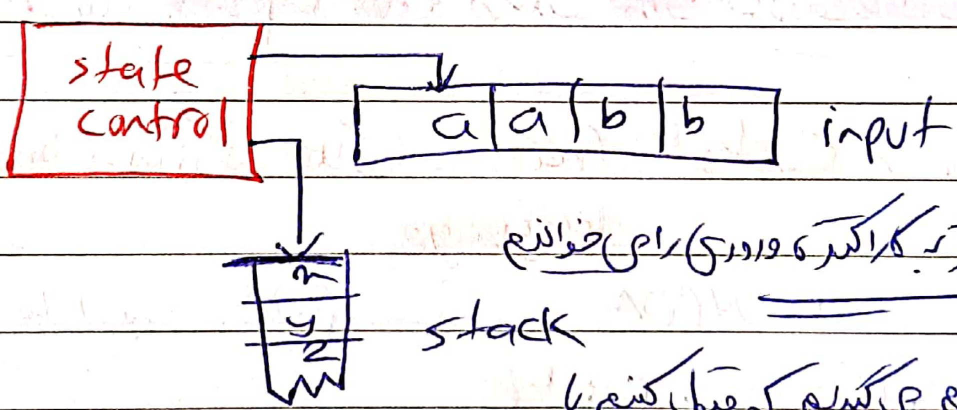
push down automata are equivalent (معادل) in power to contex free grammers. → CFG

PDA ها خیلی قوی‌تر از NFA ها هستن

shematic of a finite automaton



shematic of a pushdown automaton:



کاراکتر به کاراکتر ورودی را می‌خوانیم

و تصمیم می‌گیریم که قبول کنیم یا رد کنیم.

* می‌دانیم که توی stack همواره به عنصر بالای استک دسترسی داریم

→ push, pop

برای عنصر بالای stack

pop: برای اینکه بفهمیم توی اولین عنصر استک چی هست ←
بدون pop کردن نمی‌فهمیم چی توی عنصر اول هست ـ می‌کنیم

LIFO: last in first out

الفبایی که توی PDA داریم ← نامحدود است

✓ زبان‌هایی که finite automata ها از بیان آنها عاجز هستند

مثل $\{0^n 1^n \mid n \geq 0\}$ ← چون PDA

stack نامحدود دارد ← می‌تواند این زبان را تشخیص دهد

PDA ها → non deterministic

و

PDA های که deterministic هستند قدرت کمتری از non deterministic دارند

DFA ها و NFA ها از حیث قدرت توصیف معادل هستند

و یک کلاس (زبان‌های regular) را توصیف می‌کنند

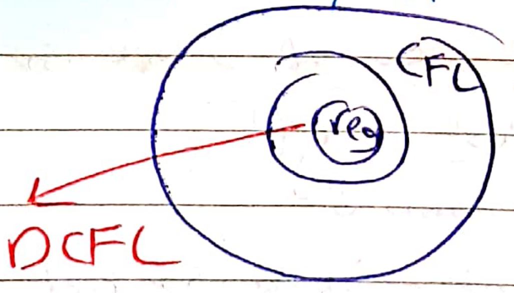
NPDA ها و DPDA ها ، از حیث قدرت توصیف متفاوت هستند

NPDA ها هستند که کلاس Context free language را توصیف می‌کنند

منظور ما از یک PDA ، NPDA است

DPDA → DCFL

NPDA = PDA ~ CFL ← CFG



(push) الفبای الک ممکن است از الفبای ورودی متمایز باشد

یعنی ممکن است کاراکترهایی که PDA ها توی stack push می‌کنند با کاراکترهایی که می‌خوانند الفبای متمایزی داشته باشند

(سیگما) الفبای ورودی $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

$\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$ الفبای stack (گاما)

برای (transition) الف

state ای که در آن هستیم هم الف ، کاراکتر ورودی هم الف ، عنصر بالای stack چی هست.

زیر مجموعه‌های از Q

همون الف برچسبا state بروم

NFA : $\delta : Q \times \Sigma_\varepsilon \longmapsto \boxed{P(Q)}$

PDA : $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow$

از ورودی چی می‌خونی

stack-top

ε ← یعنی همون الف از ورودی چیزی را نمی‌خوانه

همون الف تابع transition و بدون خواندن کاراکتری از بالای stack انجام شود

power set

خروجی تابع transition

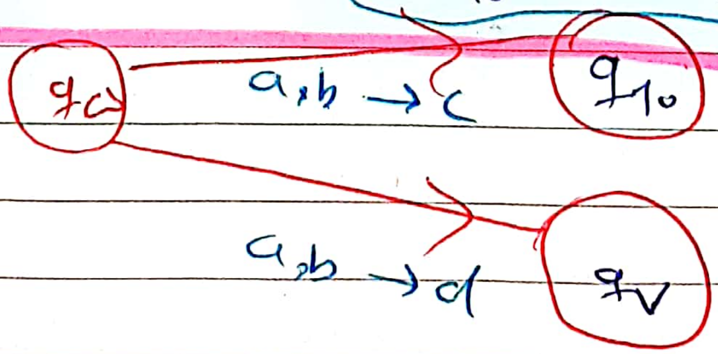
$\rightarrow P(Q \times \Gamma_\varepsilon)$

برد تابع δ برای PDA

وقتی توی q هستی و $\boxed{a}$ را داری می‌خونی و $\boxed{b}$ را از بالای stack pop می‌کنی

کاراکتری که توی stack می‌خوام push می‌کنم

$$\delta(q_0, a, b) = \{(q_{10}, c), (q_7, d)\}$$



می‌توانی بری به

۱) q_{10} و $\boxed{c}$ را توی stack push کنی

یا

۲) بری به q_7 و $\boxed{d}$ را توی stack push کنی

session 14 ادامه‌ی نظریه

PDA → transition function :

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$

power set (چون ممکن است نتیجه در دفعه، چند حالت داشته باشیم)

تعریف رسمی PDA

a push down automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Q : set of states

Σ : input alphabet الفبای ورودی

Γ : stack alphabet الفبای استک

$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ transition function

بدون خواندن از ورودی

non determinism

بدون push کردن

بدون pop کردن

$q_0 \in Q$: start state

$F \subseteq Q$: a set of accept states

مفهوم پذیرش توسط PDA

یک PDA به صورت $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

رشته‌ی w را می‌پذیرد اگر $w = w_1, w_2, \dots, w_m$ که هر $w_i \in \Sigma_\varepsilon$

و دنباله‌ای از state ها مثل

$r_0, r_1, \dots, r_m \in Q$

و دنباله‌ای از رشته‌ها به صورت

$s_0, s_1, \dots, s_m \in \Gamma^*$

وجود داشته باشد که شرایط زیر را ارضا کنند

s_i ها محتوای stack هستند

s_i محتوای استک هست بعد از خواندن هر (w_i)

شرط‌ها

start state کار را با شروع می‌کنیم

① $r_0 = q_0$ و $s_0 = \varepsilon$

در ابتدای کار استک باید خالی باشد

② for $i = 0, \dots, m-1$ we have

$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ و

$a, b \in \Gamma_\varepsilon$ و $t \in \Gamma^*$ برای $s_{i+1} = bt$

سمبلی از الفبای استک

وقتی تو state $\boxed{r_i}$ هستیم و سمبل $\boxed{w_{i+1}}$ را می‌خوانی

و $a =$ استک $-$ top

⇓

باید به r_{i+1} بروی و b را به استک push کنی

که باید $s_i = at$ و $s_{i+1} = bt$

top = a بوده

بالای استک که pop می‌کنیم

b را پوش می‌کنیم

$\boxed{t}$ ← زیر top قرار داره (ادامه‌ی استک است)

③ $\boxed{r_m \in F}$ در انتهای رشته باید اتفاق بیفتد ، accept state

---

توصیف وضعیت یک PDA حین پردازش یک رشته

توصیفات ← instantaneous description

برای یک NPDA

الان وضعیت stack و ورودی‌ها و ...

برای اینکه یک configuration از یک PDA را در یک لحظه بدانیم

با ۳ چیز مشخص می‌کنیم ← (q, w, u)

قسمت باقی‌مانده از استک

قسمت باقی‌مانده از ورودی

در آن لحظه توی چه state ای قرار دارد

یک سه‌تایی مرتب است

a config of

$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a triple (q, x, α) where $q \in Q$, $x \in \Sigma^*$, $\alpha \in \Gamma^*$

state ها — متن ورودی — محتوای استک

حرکت از یک کانفیگ به کانفیگ دیگر

را pop کردیم از — یک کاراکتر از ورودی

$(p, x, \alpha) \vdash_M (q, y, \beta)$

به ازای stack, config یک

(کلی) ε

می رویم به کانفیگ بعدی — یک config به یک config دیگر می رویم

نماد انشتاد از $\vdash$

را push کردیم

① $x = \delta y$ برای $\delta \in \Sigma$ یا $x = y$

$\Downarrow$

$x = \delta y$ برای $\delta \in \Sigma_\varepsilon$

② $\alpha = X\gamma$ $X \in \Gamma_\varepsilon$, $\gamma \in \Gamma^*$

باقی — کاراکتر

$\Downarrow$

③ $\beta = \xi\gamma$ $\xi \in \Gamma_\varepsilon$ — top زیر

قبل از top

(top زیر) (کلی) که push کردیم

$(q, \xi) \in \delta(p, \delta, X)$

از رشته ورودی $\boxed{\delta}$ را می خوانیم — چیزی که از بالای استک می خوانیم

به state q می رویم — بالای استک این کاراکتر را push می کنیم

رشته ورودی — بعدی state

$(p, x, \alpha) \vdash (q, y, \beta)$ — المان stack

state فعلی — محتوای استک — ادامه ی رشته ی ورودی

الم دلتاک

$(p, x, \alpha) \vdash^{n}_{m} (q, y, \beta)$

با $\boxed{n}$ تا گام از

رفتیم به

config مبدا

فاز دوم β

$(p, x, \alpha) \vdash^{*}_{m} (q, y, \beta)$

طی صفر یا بیشتر گام،

از مبدا کانفیگ دوم رفتیم

به رالیک

فاز سوم β

مفهوم پذیرش توسط PDA

PDA ـ m ، رشته‌ی $\boxed{x}$ را قبول می‌کند اگر

$(q_0, x, \varepsilon) \vdash^{*}_{m} (q, \varepsilon, \alpha)$

① استک خالی باشد

② رشته‌ی ورودی $= x$

③ start state $= q_0$

① به یک q برسیم که $q \in F$ — accept state — بدله — برقرار باشد

② رشته را کامل خوانده باشیم طی صفر یا بیشتر گام

③ محتوای استک دیگر مهم نیست

* برای اینکه m ، رشته‌ی x را قبول کند باید کاملاً x را بخواند

چیزی از x نماند

$\alpha \in \Gamma^{*}$ ، $q \in F$

اگر قبول کرد

$L = L(m)$

($ ← کف stack را نشان می‌دهد)

session 15 اثر انی ، DCFL , DPDA

اثر ان determinism

$\boxed{ex1}$ the following is the formal description of an NPDA that recognize the language $\{0^n 1^n \mid n \geq 0\}$. Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ where $Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$

and $\boxed{\delta}$ is given by following table

(جدول تابع transition را توصیف می‌کند)

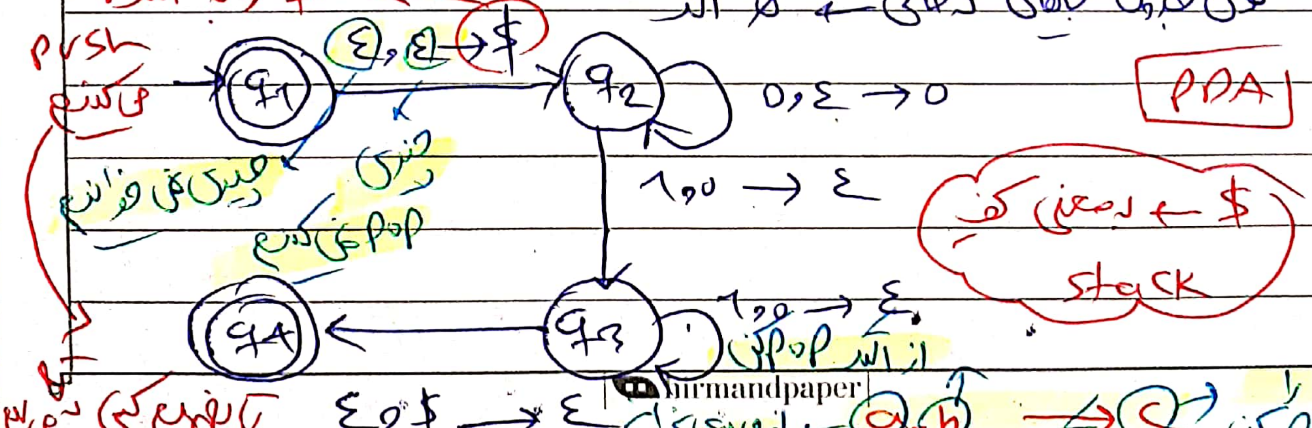
input	0			1			ε		
stack	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

اگر از الفبا چیزی نخوانیم ، از الفبا 0 را

$\delta(q_2, 0, \varepsilon) = \{(q_2, 0)\}$ ← top stack صفر را

state در ای ، این را می‌خوانیم از ورودی ، در q_2 می‌مانیم ، اضافه می‌کنیم (push می‌کنیم)

$ را به الفبا ، توی جدول جاهای که خالی ← $\emptyset$ الفبا



تا خالی کنیم

$\varepsilon, \$ \to \varepsilon$ ، از ورودی کف a ← b ، c را push کن تو الفبا

طبق نمودار توی $\boxed{q_2}$ که هستیم ← همه‌ی صفرها را از ورودی می‌خوانیم و
صفر توی استک push می‌کنیم

بعد ورود می‌کنه به $\boxed{1}$ خوندن و

به ازای هر $\boxed{1}$ که می‌خونه ← یکی از صفرهای داخل استک
را حذف می‌کنه (pop می‌کنه)

بعد که همه‌ی $\boxed{1}$ ها را با $\boxed{0}$ ها زد

اگه یک $ دید ← میره به stack $\boxed{q_4}$

تمام می‌شود

اونی که می‌گه تا زنجیره‌ای
$\boxed{0011}$ را قبول کنه.

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$) \vdash (q_2, 011, 0\$)$$
$$\vdash (q_2, 11, 00\$) \vdash (q_3, 1, 0\$) \vdash (q_3, \varepsilon, \$)$$
$$\vdash (q_4, \varepsilon, \varepsilon)$$

1 را با 0 می‌زند

1 را با 0 ها می‌زند

به ازای الف $ تموم

2 و 3 ← صفرها را از ورودی می‌خونه و توی استک می‌ریزه

4 و 5 و 6 ← 1 ها را ... با صفرهای که تو استک هست
می‌زند

* برای PDA ها نوع درخت تجزیه داریم

اگر حداقل یکی از شاخه های درخت من را به یک accept state برساند ← PDA آن رشته را قبول می کند

کانفیگ ابتدایی

011

$(q_1, 011, \varepsilon)$ هم زمان نمی توانیم بخوانیم ... وضعیت قبل از خواندن صفر اول ← $(q_2, 011, \$)$

نقطه صفر خروجی ندارد ✗ ... ۰

$(q_2, 11, 0\$)$

۱

هم زمان نمی توانیم بخوانیم $(q_3, 1, 0\$)$ ... $(q_4, 1, \varepsilon)$

۱ ✗ ... ✗

$\delta(q_3, 1, \$) = \emptyset$

نقطه ۱ خروجی ندارد ← این راه هم بسته می شود

$(q_3, 1, \$)$ چون بالای استک $ است ... نمی توانیم ... بعد از حذف ... دو

$(q_4, 1, \varepsilon)$ برویم توی

پس برای این رشته ← بعد از اینکه کل رشته را خواندیم هیچ کدام از شاخه ها نتوانست من را به یک final state برساند

قبول می شود

0011

نقطه صفر خروجی ندارد ✗ $(q_1, 0011, \varepsilon)$ ... $(q_2, 011, \$)$

۰

کانفیگ ابتدایی ← از بالا

$(\varepsilon, \varepsilon \to \$)$ انتقال می کنیم و می رویم توی حالت بعدی

$(q_2, 011, 0\$)$

۰

$\delta(q_2, 1, 0) = \{(q_3, \varepsilon)\}$ ... $(q_2, 11, 00\$)$

۱

می توانیم همزمان ۲ تا کانفیگ را انجام دهیم.

وقتی توی این هستیم و ۱ را می خوانیم ← $(q_3, 1, 0\$)$

۱

$(q_3, \varepsilon, \$)$ می رویم توی

چون بالای استک $ داریم ← می توانیم ... $(q_3, \varepsilon, \$)$ ... $(q_4, \varepsilon, \varepsilon)$

برویم توی state بعدی $\delta(q_3, \varepsilon, \$) = \{(q_4, \varepsilon)\}$

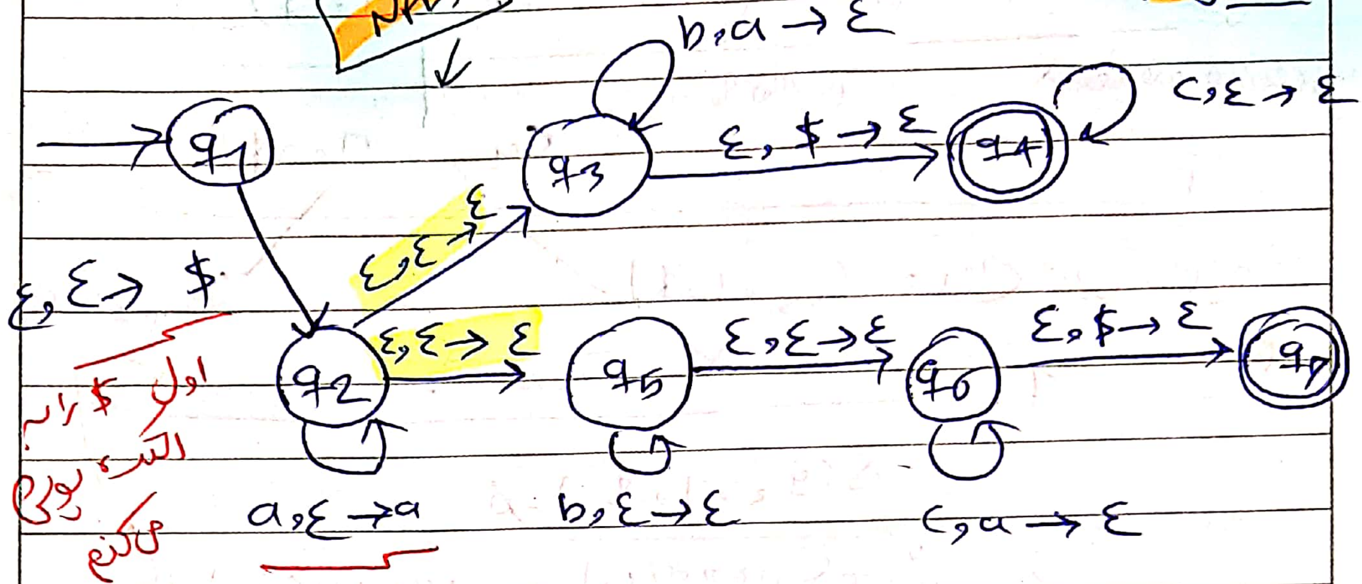
را قبول می کند $\boxed{0011}$ ← این PDA ← final state

... یعنی هم که state ... $(q_4, \varepsilon, \varepsilon)$ است ← قبول می شود

(DCFL نیست) الف CFL ← این زبان non determinism الف

ex2) this example illustrates a PDA that recognizes the language $\{a^i b^j c^k \mid i,j,k \geq 0$ و $i=j$ or $i=k\}$

NPDA



اول $ را به استک پوش می کنم

دوم به تعداد a ها را می خوانم و push می کنم توی استک

سوم ← دو حالت داریم ← تعداد a و b برابر باشد
تعداد a و c برابر باشد

برای حالت b ← حالت 3 به تعداد b های که از ورودی می خوانیم یک a را از استک pop می کنیم

بعد از خواندن همه ی b ها وقتی استک خالی شد و به $ رسیدیم

$

هر چقدر c هست را می خوانیم و به accept رسیدیم

چون مطمئن شدیم تعداد a و b برابره

برای حالت c ← حالت 5 هر چقدر می خوای b بخوان و بدون دست زدن محتوای استک

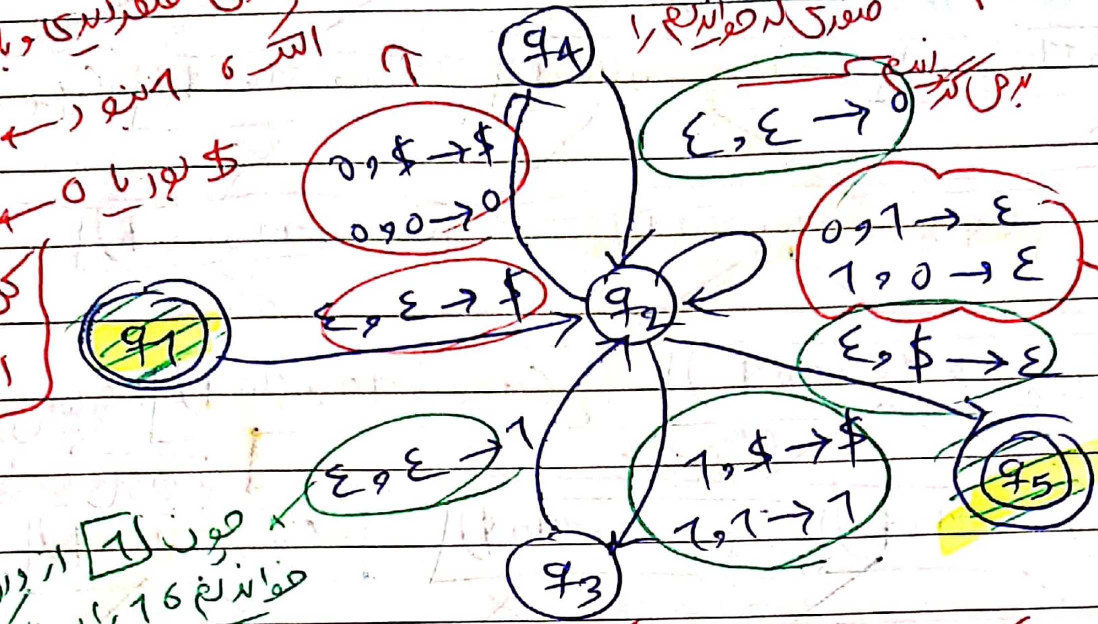
پنجم 6 به تعداد a های که ذخیره شده در استک c می خوانیم

و در حالت وقتی استک به $ رسید ← به accept می رسیم

ex3 $\{w \in \{0,1\}^* \mid n_0(w) = n_1(w)\}$

تعداد صفر و یک برابر

طراحی یک PDA



اولین کار یک $\boxed{\$}$ توی استک push می‌کنم که نشان بدهد زیر استک کجاست

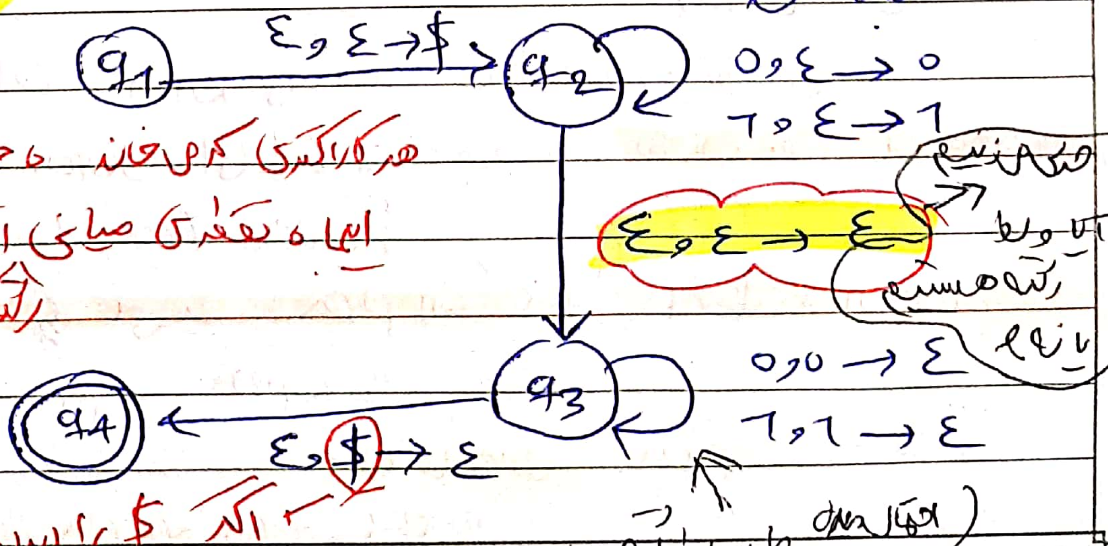
اگر از ورودی صفر خواندی و بالای استک $\boxed{1}$ بود ← با هم بزن

یک خواندی و بالای استک $\boxed{0}$ بود ← ...

اگر تعداد صفرها و یک‌ها برابر بودند ← استک خالی می‌شود ← به استیت final می‌رسیم

ex4 PDA ای که زبان $\{ww^R \mid w \in \{0,1\}^*\}$ را نشان بدهد

NPDA لازم داریم



هر کاراکتری که می‌خواند حدس می‌زند احتمالاً اینجا نقطه‌ی میانی است

وارد q3 می‌شود pop می‌شود

اگر به $ رسیدیم final

احتمالاً ما به وسط رشته رسیدیم

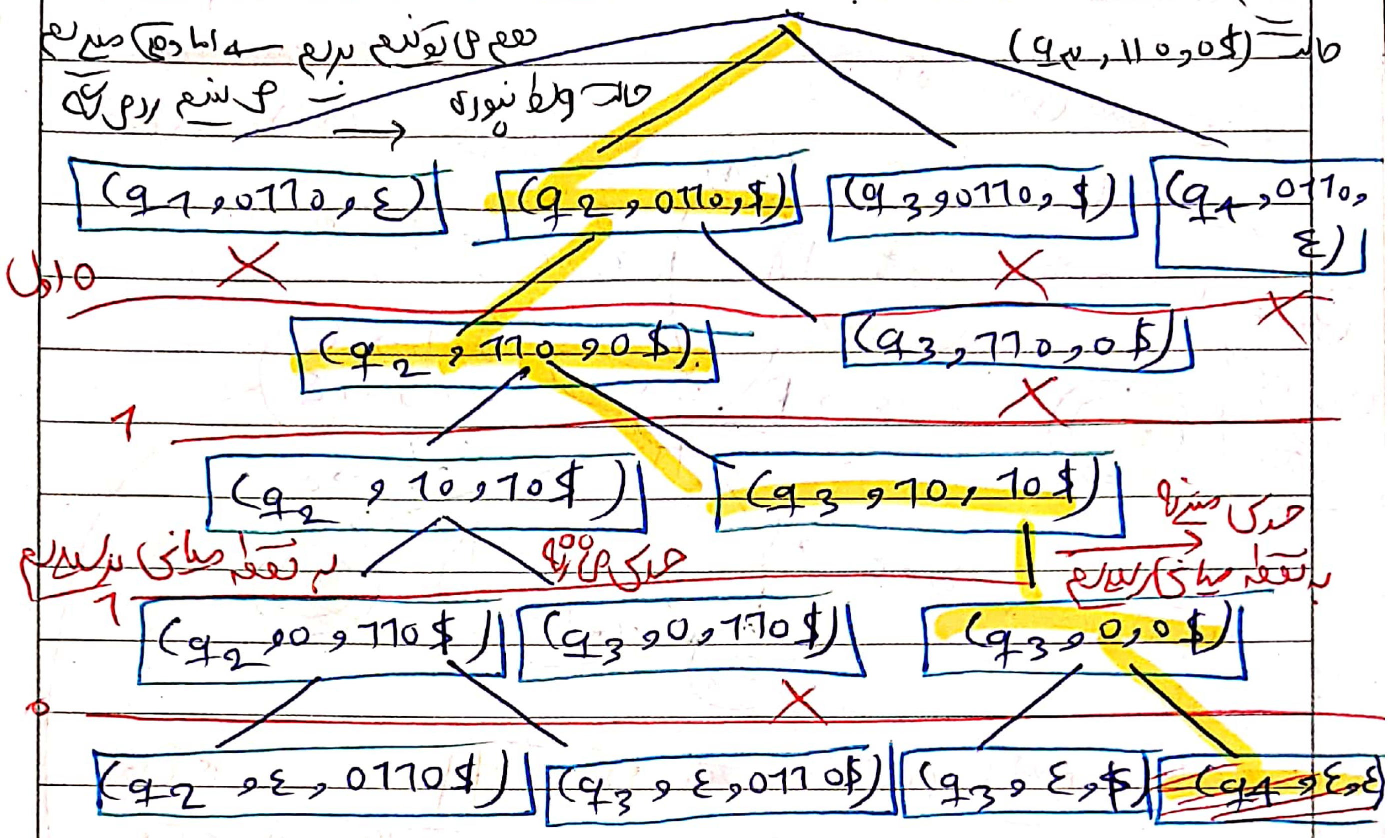
هر باری حرکت جدید می‌زنه آیا به نقطه‌ی پایانی رسیده یا نه؟

صفر، ۲ و صفر توی

$(q_2, 0110, \$)$ هستیم ← فرض می‌کنیم به نقطه کاری

الان هم توی درخت محاسبه رشته 0110 ε

حالت $(q_2, 110, 0\$)$



قبل از اینکه اولین صفر را بخوانیم state ε رو می‌تونیم داشته باشیم

CFG ها و DPDA ← are equivalent in power

معادل ← NPDA ها

* یک زبان مثل L زمانی DCFL است اگر و تنها اگر یک DPDA مثل M وجود داشته باشد که

$L = L(M)$

تفاوت DFA و NFA می‌شه یک CFG طراحی کرد ←

هر زبان regular ای CFL هم هست — هر زبان منظمی CFL هم هست

context free language

FA

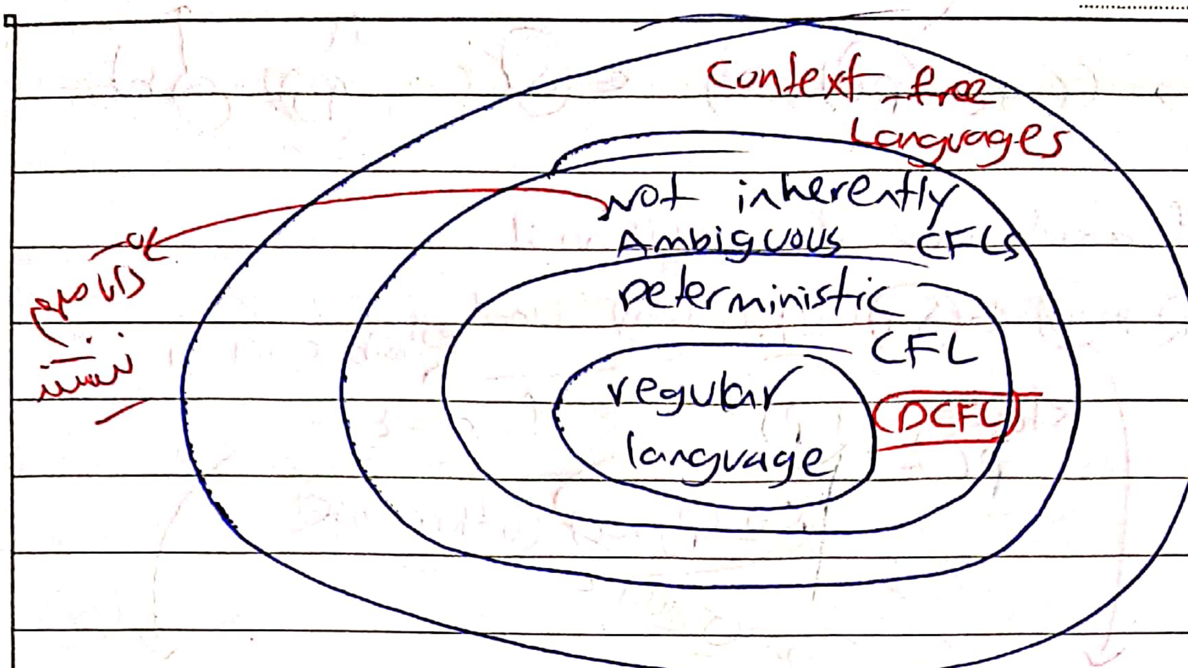
هر NFA ای یک PDA است که از stack استفاده می‌کند

finite automata

قدرت PDA و NPDA ها معادل نیست ← NPDA که غیرقطعی

است قدرتمندتر است — CFG ها معادل NPDA ها هستند

زبان تعریف NPDA ها CFL ها



نظریه ۶: اگر L_1 یک CFL باشد و L_2 یک regular language باشد $\rightarrow$ $\boxed{L_1 \cap L_2}$ یک Context free است

دیگه تقلید اشتراک یک PDA را Context free است

اثبات: ۶ فرض کنیم — اثبات کردیم که اشتراک ۲ CFL است، CFL است

$M_1 = (Q, \Sigma, \Gamma, \delta_1, q_0, F_1) \rightarrow$ NPDA

است که L_1 را قبول می‌کند

$M_2 = (P, \Sigma, \delta_2, p_0, F_2) \rightarrow$ DFA

که L_2 را قبول می‌کند

یک PDA تقلید اشتراک

$\hat{M} = (\hat{Q}, \Sigma, \Gamma, \hat{\delta}, \hat{q}_0, \hat{F})$

هم استار M_1 را و هم استار M_2 را شبیه‌سازی می‌کند

$\hat{Q} = Q \times P$ ①

$\hat{q}_0 = (q_0, p_0)$ ② → start M_1, M_2 start

$\hat{F} = F_1 \times F_2$ ③ ← ضرب دکارتی F_1, F_2

هر رشته‌هایی به تعریف اشتراک باشد (هم M_1 را قبول می‌کند هم M_2)

باید $\hat{\delta}$ (تابع transition) را تعریف کنیم

$\in \Sigma_\varepsilon$ اگه توی زبانم و a را بخوانم ← به p_L می‌رم

$\hat{\delta} \rightarrow ((q_k, p_L), n) \in \hat{\delta}((q_i, p_j), a, b)$

if and only if از ورودی a

$(q_k, n) \in \delta_1(q_i, a, b)$ and از الفبا b بخوانم

برای state جدید

n را به الفبا push می‌کنم

state $p_L = \begin{cases} p_j & \text{if } a = \varepsilon \\ \delta_2(p_j, a) & \text{otherwise} \end{cases}$

در غیر این صورت

اگر از ورودی چیزی نخوانی

نباید توی DFA حرکت کنی

$\boxed{p_j = p_L}$

پس توی M_2 هیچ اتفاقی نمی‌افتد

→ simulate کن

اگر تو state p_j باشم و $\boxed{a}$ را بخوانم به p_L می‌رم

(رشته ورودی)

$((q_0, p_0), w, \varepsilon) \vdash_{\hat{M}} ((q_r, p_s), \varepsilon, n)$ پس اگر

$q_r \in F_1$ و $p_s \in F_2$

($\hat{M}$ در accept state یک)

اگر $\boxed{w}$ تمام کند

① $(q_0, w, \varepsilon) \vdash_{M_1} (q_r, \varepsilon, n)$

موقع accept

برای M_1 ←

و

② $\delta^*(p_0, w) = p_s$ ← برای M_2

در M_2 هم توسط یک موقع accept می‌کنم

زبانی که توسط $\hat{M}$ recognize می‌شه اشتراک هست که زبانش می‌کردیم

$\hookrightarrow L(\hat{M}) = L_1 \cap L_2$ → یک طراحی PDA کردیم

← CFL الد

اشتراک ۲ تا CFL ← لزوماً CFL نمی‌شه

تمرین: $L = \{a^n b^m c^{n+m} \mid n \geq 0, m \geq 0\}$

یک PDA طراحی کن

Session 15 — Pumping lemma for context-free languages

اگر A یک CFL باشد ← یک p (pumping length) وجود دارد که اگر s یک رشته در A باشد و طولش حداقل p باشد ← s می‌تواند به ۵ قسمت تقسیم شود ← همه‌شون توی A

$s = uvxyz$ ← که باید ۳ شرط را ارضا کند

① for each $i \geq 0$, $uv^ixy^iz \in A$

② $|vy| > 0$ و $vy \neq \varepsilon$

③ $|vxy| \leq p$

v و y به همراه هم ε نیستند

$L(G) = A$

این رشته به ازای هر $i \geq 0$ ← داخل زبان هستند

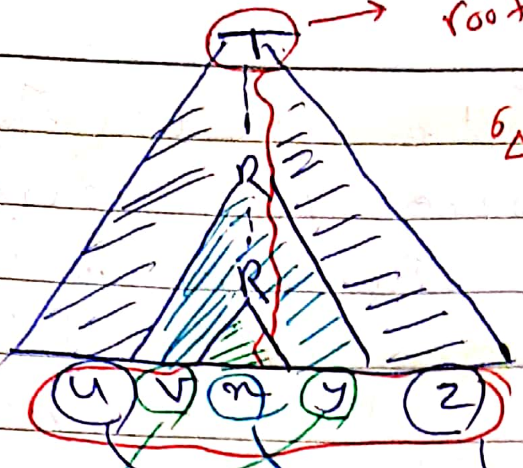
اثبات: اگر A یک CFL باشد و G هم یک CFG باشد که A را تولید می‌کند ← ما باید نشان بدیم که هر رشته‌ی طولانی مثل s که در زبان A هستند می‌توانند pump شوند و در A باقی بمانند.

اگر s یک رشته‌ی خیلی بلند در A باشد ← چون s توی A هست، G می‌تواند این s را بسازد ← یک parse tree دارد ←

این درخت برای s باید یک درخت خیلی بلند باشد چون s خیلی بزرگ است. بلندترین مسیر درخت یعنی یک مسیری وجود دارد از start که در ریشه هست به یکی از ترمینال‌هایی که در برگ قرار می‌گیرد

حالا چون مسیر خیلی طولانیه ← یک متغیر داریم مثل R که R دوبار در مسیر ظاهر شده، همینا که R را پیدا می‌کنیم که دوبار در مسیر ظاهر شده، چیزی که می‌خواهیم را به ما می‌دهد

root ≡ start متغیر



$R \in V$ یک متغیر است که توی مسیر دوبار ظاهر میشه

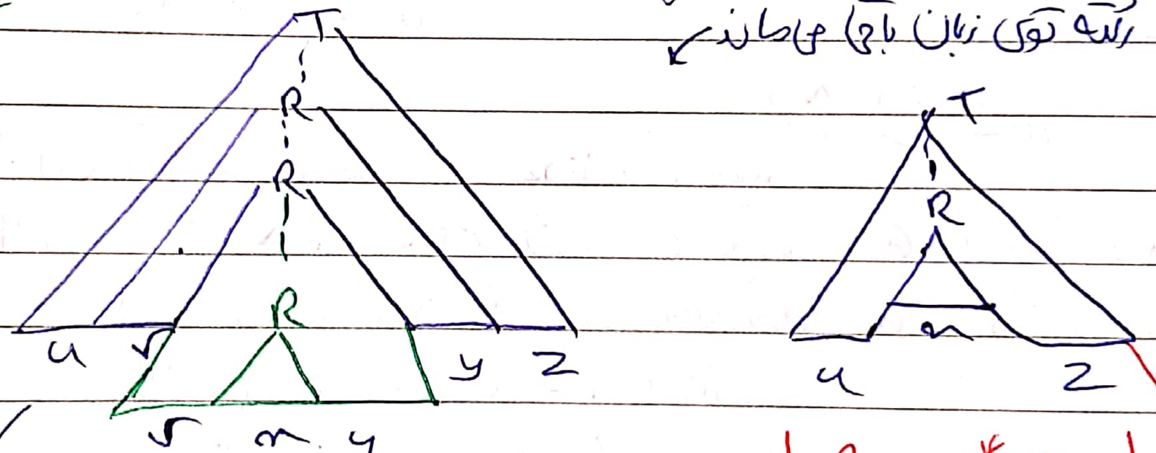
رشته‌ای که الان که داره ساخته میشه

توی sub_tree R بالایی هست (توی پایینی نیست)

توی sub_tree R پایینی (توی بالایی هست)

توی هیچ کدام از sub_tree های [R] ها نیست

اینه حالا من سیم‌های دوم و چهارم را حذف می‌کنم یا اضافه می‌کنم و رشته توی زبان باقی می‌مانند



$R \Rightarrow^* vxy$

$R \Rightarrow x$

R می‌تواند vxy را بسازد

x را هم بسازد

به بالایی R که رسیدیم به جای اینکه vxy بسازی، فقط x بساز

به وقتی پایینی تر R که رسیدیم به جای ساخت x، vxy بساز

→ uv^2xy^2z

داریم

برای uv^3xy^3z به R آخری می‌رسیم به جای ساخت x، $\boxed{vxy}$ بساز

حالا باید تعیین کنیم $\boxed{p}$ چی باشه تا تضمین داشته باشیم CR دوباره رخ می‌دهد

و باید یه کرانه را اثبات کنیم $\boxed{|s| \geq p}$ CFG تفسیر A

T

اصل اثبات ← B ← طولِ طولانی‌ترین rule در گرامر G

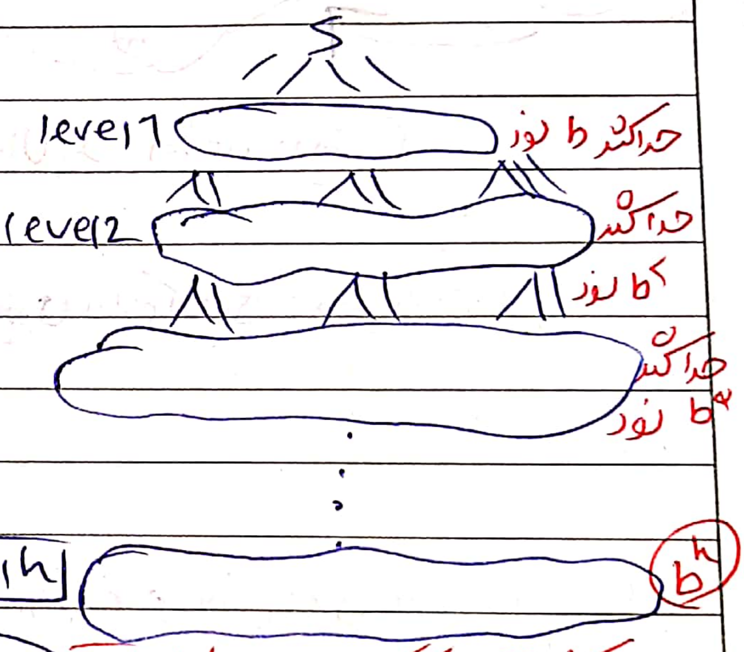
b

$\boxed{b \geq 2 \text{ الف}}$ فرض می‌کنیم

د) اگر طولِ طولانی‌ترین برابر ۱ باشه یا کمتر ← زبان متناهی میشود

lem ← پامپینگ به صورت بدیهی

برقرار می‌شود



b^h+1 ← اگر یک رشته به طول

داشته باشیم ← ارتفاع درخت

تجزیه‌ی اون رشته

باید حداقل $h+1$

الا

حداکثر تعداد برگ ← b^h الف

$$p = b^{|V|+1}$$

$|s| \geq b^{|V|+1}$ ← $|s| \geq p = b^{|V|+1}$

$\gg b^{|V|}+1$

ارتفاع درخت تجزیه‌ی حداقل $|V|+1$

J یک درخت تجزیه‌ی s الف با کمترین تعداد نودها

← ارتفاع این J حداقل $b^{|V|}+1$ الف

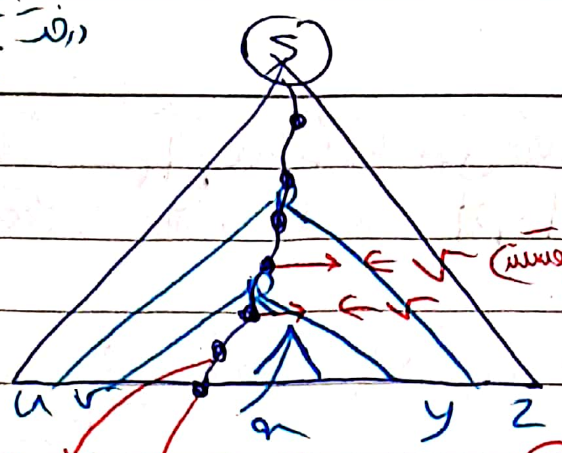
$|s| \geq p = b^{|V|+1} \gg b^{|V|}+1$

ارتفاع درخت تجزیه‌ی s ←

است $(|V|+1) = \boxed{\text{حداقل}}$

درخت تجزیه تولید S

J



طول این مسیر حداقل $|V|+1$

حداقل $|V|+2$ تا نود

حداقل $|V|+1$ تا نود تولید متغیرهای G در این مسیر ظاهر میشود

آخری لازم اون ترمینال الف (متغیر نیست)

طبق اصل لانه کبوتری ۲ تا متغیر داریم که توی این مسیر داره تکرار میشه

این همون متغیر R است

این مسیر را از پایین به بالا میرویم و میبینیم اولین تکراری هست R

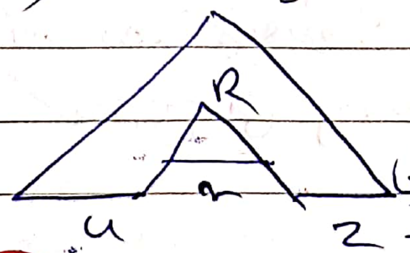
اسمش را R میگذاریم

تجزیه را طبق دیاگرام های قبل مشخص میکنیم

حالا شرط ها را چک میکنیم

① $uv^ixy^iz \in A \quad \forall i \geq 0$ ✓

② فرض کن vy دو تاش تهی باشند، درخت تجزیه‌ای خواهیم داشت که از $\boxed{J}$ کوچکتر میشه



این با این که گفتیم که $\boxed{J}$ کوچکترین درخت الف هست در تناقض است

$|vxy| \leq p$ ✓

یا v باید ناتهی باشد یا y یا هر دو.

③ نکته ← $|vxy| \le b^{|V|+1}$ ←

$|vxy| \le p$ ← p الف

چون R را

طوری انتخاب کردیم که از پایین درخت که حرکت به بالا هستیم R اولین تکرار کنیم

↓

R توی $|V|+1$ متغیر پایین مسیر است

↓ زیر درختی که R ، vxy را تولید می کند حداکثر $\boxed{|V|+1}$ ارتفاع دارد → یک درخت با این ارتفاع به رشته به طول حداکثر $b^{|V|+1} = p$ را می تواند تولید کند

ex1 Use the pumping lemma to show that the language $B = \{a^n b^n c^n \mid n \ge 0\}$ is not context free.

اول فرض می کنیم (فرض خلف) که زبان B یک CFL است و فرض می کنیم p همان pumping length باشد که لم تضمین برای ما تعیین می کند و فرض می کنیم s رشته ای باشد که داخل زبان B است و به فرم $s = a^p b^p c^p$ است و $|s| = 3p > p$ است به طوری که لم تضمین کند که برای s باید یک decomposition برای s معرفی کند طوری که ۳ شرط لم را ارضا کند چون فرض کردیم B یک CFL است پس باید یک تجزیه برای s معرفی کند که این تجزیه ۳ شرط لم را ارضا کند (که به $s = uvxyz$ تجزیه می شود) شرط ۳ می گه که $\boxed{|vxy| \le p}$ است

پس vxy در s نمی تواند هم حاوی a باشد هم حاوی c

$\underbrace{a \cdots a}_{p}\ \underbrace{b \cdots b}_{p}\ \underbrace{c \cdots c}_{p}$

اگر vxy بخواهد شامل a و c باشد ← طول آن بیشتر از p می شود ✗

پس vxy فقط یا شامل a و b باشد یا b و c

پس vxy یا ① زیررشته‌ای از a^pb^p است ← چون طول ≤ p

② ... b^pc^p ... پس که

$vy \neq \epsilon$ است

شامل a یا b است ← یا v یا y دارای a است

اگر رشته‌ی uvxyz را صفر بار پمپ کنیم ←

uv^0xy^0z

اینجا تعداد c های دقیقاً برابر p است ولی

تعداد a ها یا تعداد b ها یا هردو اکیداً کمتر از p است ← پس رشته‌ی

حاصل uxz داخل زبان نیست

برای حالت دوم ② اگر های رشته‌ی uv^0xy^0z را در نظر بگیریم ←

تعداد a ها دقیقاً برابر p است ولی تعداد b ها یا c ها یا هردو اکیداً کمتر

از p است ← $uxz \notin B$

تناقض با فرض خلف ←

فرض خلف باطل ← CFL نیست (!)

ex2: $L=\{a^ib^jc^k \mid k=\max\{i,j\}\}$ is not CFL

تعداد c ها = ماکس تعداد a ها و b ها

فرض خلف می‌کنیم که L یک CFL است و فرض می‌کنیم p طول پمپینگ باشد

که توی لم تزریق‌ها، وجود داره گفته شده؛ یک رشته‌ای مثل s در نظر می‌گیریم که

باید طولش از p بیشتر باشد ← $s=a^pb^pc^p$

میشه s را به شکل uvxyz ← $|s|=3p>p$

compose کرد که ۳ شرط لم را ارضا کنند ← طبق شرط ۳

$|vxy|\le p$ است ← این رشته نمی‌تواند هم حاوی a باشد هم c ←

۲ حالت داریم case1: vy شامل b باشد ← چون طولی

قراره پمپ بشه این دوتا مهم هستند ← حالت‌ها را روی vy می‌گذاریم

چون vy حاوی b است ← دقیقاً در vxy a نداریم

اگر صفر بار پمپ کنیم ← uxz ← تعداد a ها در uxz

دقیقاً برابر p است اما تعداد c ها 2 اکیداً کمتر از p است ← اما $uxz \notin$

داخل زبان قرار نمی‌گیرد — تعداد a ها $= p$

چون در زبان داریم ← تعداد c ها $> p$

تعداد c ها باید ماکس تعداد a و b باشد ← تناقض داره X

case 2: vxy حاوی c نباشد ← میایم رشته‌ی s را دوبار پمپ می‌کنیم تا رشته‌ی $\boxed{uv^2xy^2z}$ ایجاد شود ← pump up

تعداد c ها = دقیقاً برابر p است

و تعداد a یا b ها 2 اکیداً از p بیشتر است و ...

پس دوباره فرم زبان را از دست می‌دهد ← X داخل زبان نیست $\boxed{uv^2xy^2z \notin L}$

پس فرض خلف باطل ← زبان L، CFL نیست

ex3 $C = \{a^i b^j c^k \mid 0 \le i \le j \le k\}$ مثلاً $aabbccc$

فرض خلف می‌کنیم C یک CFL است ، p طول لم تزریق است و s یک رشته‌ای به فرم $s = a^p b^p c^p$ است که $|s| = 3p \ge p$ است و داخل زبان است ، طبق لم تزریق باید s بتواند decompose شود طوری که به فرم $uvxyz$ شروط لم تزریق را ارضا کند

طبق شرط سوم $\boxed{|vxy| \le p}$ است ← vxy نمی‌تواند هم a داشته باشد هم c ← ۲ حالت داریم

case 1: v یا y شامل c باشد ← یک pump down انجام می‌دهیم یعنی uv^0xy^0z را در نظر می‌گیریم ← در رشته‌ی uxz چون v یا y شامل c بود ← تعداد c ها اکیداً از p کمتر است و تعداد a ها 2 دقیقاً برابر p است

$$\boxed{\begin{array}{l} n_a(uxz) = p \\ n_c(uxz) < p \end{array}} \rightarrow \boxed{uxz \notin C}$$

پس برای حالت بعدی میایم

$\boxed{s' = uv^2xy^2z}$

case 2: v یا y شامل c نباشد ← اینجا میایم pump up می‌کنیم یعنی $\boxed{uv^2xy^2z}$ را در نظر می‌گیریم

چون v یا y، c ندارد ← در رشته‌ی uv^2xy^2z ، تعداد a ها یا b ها اکیداً از $\boxed{p}$ بیشتر است و تعداد c ها در s' دقیقاً برابر p است

تناقض با فرم زبان ← s' داخل زبان نیست ← فرض خلف باطل ← C CFL نیست

$n_c(s') = p$, $n_a(s') > p$ or $n_b(s') > p$ → به فرم زبان نیست

ex: $L = \{w \in \{a,b,c\}^* \mid n_a(w) < n_b(w)$ and $n_a(w) < n_c(w)\}$

همه رشته‌هایی که تعداد a ها اکیداً کمتر تعداد b ها و تعداد a ها اکیداً کمتر تعداد c هاست

مثلاً abbccc یا ccabb

فرض می‌کنیم L یک CFL است (فرض خلف) و p یک عدد صحیح است که لم پمپینگ به ما می‌دهد، یک رشته‌ای مثل $s = a^p b^{p+1} c^{p+1}$ وجود دارد

که چون طولش بیشتر از p است و داخل زبان است ← می‌شه تجزیه‌اش کرد

به فرم uvxyz طوری که ۳ شرط لم را ارضا کند

case 1: اگر vy شامل a ها باشد ← نمی‌تواند c داشته باشد چون $\widehat{vxy}$

طبق شرط سوم $|vxy| \le p$ است ← اینجا پمپ آپ (pump up) می‌کنیم ←

در رشته uv^2xy^2z تعداد a ها حداقل $(p+1)$ است

$n_a(s') \geq p+1$ ← $s' = uv^2xy^2z$

$n_c(s') = p+1$ ولی تعداد c ها دقیقاً برابر p+1 است

تناقض

رشته تولید شده در فرم زبان نیست

case 2: اگر vy شامل a نباشد pump down می‌کنیم

یعنی $s'' = uv^0xy^0z$ ← $s'' = uxz$ تعداد a های s''

دقیقاً برابر p است و در s'' یا تعداد b ها کم میشه یا c ها ←

$n_a(s'') = p$

$n_b(s'') < p+1$ یا $n_c(s'') < p+1$ → تناقض داریم

$n_b(s'') \le p$ یا $n_c(s'') \le p$ با فرم زبان

پس فرض خلف باطل و L یک CFL نیست

ex51 $L = \{a^p b^n c^p d^n \mid p, n \geq 6\}$ is not CFL

فرض خلف می‌کنیم که L یک CFL است ← طبق لم تزریق یک عدد طبیعی مثل p داریم و یک رشته‌ای مثل $s = a^p b^p c^p d^p$ که هم طولش بیشتر از p است هم داخل زبان است ← می‌توان s را طوری تجزیه کرد که به فرم uvxyz دربیاید و ۳ شرط لم تزریق را ارضا کند

$s = uvxyz$

طبق شرط دوم لم ← $|vxy| \leq p$ باید باشد ← vxy باید ذیل a یا b یا c یا d (دقیقاً یک کاراکتر) قرار بگیرد ← ۴ تا حالت داریم

مثلاً فرض کنیم رشته‌ی vxy ما فقط شامل a باشد ←

اگر pump down کنیم و رشته‌ی uv^0xy^0z را در نظر بگیریم در این رشته تعداد a ها اکیداً کمتر از p است در حالی که تعداد c ها دقیقاً برابر p است ← رشته حاصل در زبان نمی‌افتد ← در این ۴ حالت شرط نقض می‌شه

یا اینکه رشته‌ی vxy ذیل دو تا کاراکتر مجاور است ← ذیل ab یا bc یا cd باشد ← اینجا هم ۳ حالت داریم

$\boxed{a^p b^p \text{ ذیل}}$ یا $\boxed{b^p c^p}$ یا $\boxed{c^p d^p}$

زیررشته‌ای از

مثلاً اگر ذیل $a^p b^p$ باشد

vxy زیررشته‌ای از $a^p b^p$ باشد اگر pump down انجام دهیم ←

رشته uxz ایجاد می‌شه ← در این رشته $p = n_c(s') = n_d(s')$

یعنی تعداد c و d ها تغییری نمی‌کنه

اما $n_a(s') < p$ or $n_b(s') < p$ است ← در این صورت یا

$n_a(s') \neq n_c(s')$ یا $n_b(s') \neq n_d(s')$

← فرض خلف باطل ← $\boxed{uxz \notin L}$ و دیگر زبان رابسته نمی‌شود

زبان CFL نیست

ex6 $L = \{ww^Rw \mid w \in \{0,1\}^*\}$

فرض می‌کنیم L یک CFL است و p عددی است که لم تزریق از آن وجود دارد و
$|s| \ge p$ و s داخل زبان است اما به دلیل $s = 0^p10^p1$

نمی‌خواهد چون باید نشان دهیم به تناقض رسید یعنی این تجزیه‌ای که ما دنبالش هستیم
که بگیم وجود ندارد و برای این رشته s وجود دارد ← این رشته مثال نیست
برای اینکه ما را به تناقض برساند

$$\underbrace{000 \ldots 000}_{0^p}\ 0\ 1\ \underbrace{0\ 000 \ldots 0001}_{0^p1}$$ تجزیه s

u ، v ، x ، y ، z

p-1 تا صفر

p-1 تا صفر و یک

این تجزیه‌های ۳ شرط لم را دارد $|vxy| \le p$ و $|vy| > 0$ و
uv^ixy^iz هم در زبان می‌افتد ← تجزیه‌ای که صفحه قبل هر ۳ شرط را ارضا
می‌کند ← تناقضی نداریم ← X این رشته مثال نیست

$s = 0^p1^p0^p1^p$ را در نظر می‌گیریم ←

$|vxy| \le p$ است طبق شرط ۳ ← ۳ تا case داریم که

$0^p\ 1^p\ 0^p\ 1^p$

پس vxy یا در 0^p1^p است یا 1^p0^p یا 0^p1^p

اگر pump down کنیم یعنی $i=0$ ، uxz را در نظر بگیریم و
مثلاً برای حالت اول اگر ~~[illegible]~~ s ذیل 0^p1^p باشد ←
یا تعداد صفرها کمتر می‌شود یا تعداد ۱ها ← فرم www به هم می‌ریزد ←
تناقض ایجاد می‌شود (دیگر آن رشته‌ها به هم نمی‌خورند)
مثلاً رشته از 000111 000111 بشه
تعداد صفرهای کم شده مثلاً 00111 000111

CFL زبان نیست

اگر یک حالتی پیدا بشه که نشه تونست به تناقض رسید ← این استدلال
کلاً معتبر نیست

ex 1 $L = \{n \in \{1\}^* \mid \text{length of } n \text{ is a prime}\}$

طول رشته‌ای که اول است — is not CFL.

فرض خلف می‌کنیم که این زبان CFL است. اگر p همان عددی باشد که لم تزریق وعده‌ی وجودش را می‌دهد، رشته‌ی $s = 1^\alpha \in L$ را در نظر می‌گیریم که α یک عدد اول بزرگتر از $\boxed{p+2}$ است. فرض کنید $s = uvxyz$ همان تجزیه‌ای باشد که لم وجودش را تضمین می‌کند.

$$|uv^0xy^0z| = |uxz| = \beta$$

طول uxz را β می‌گیریم

توان پمپ یا تعداد تکرار را برابر β می‌گیریم ←

$$|uv^\beta xy^\beta z| = |uxz| + \beta|vy| = |uxz| + \beta(|s| - |uxz|)$$

$$= \boxed{\beta + \beta(\alpha - \beta)} = \beta(1+\alpha-\beta)$$

چون $|vy| = |s| - |uxz|$

$\beta(1+\alpha-\beta) = \boxed{\text{یک عدد مرکب است}} \rightarrow$ چرا

باید نشان بدیم طول این دو عددی که در هم ضرب شدن ≥ 2 است

(1) $\beta = |uxz| = |s| - |vy| \geq p+2 - |vy| \geq p+2 - |vxy| \geq 2$

چون گفتیم α عددی اول بزرگتر از p+2 است

$|vy| \leq |vxy|$

طبق لم داریم $|vxy| \leq p$ است

$p+2 - \underbrace{|vxy|}_{\leq p} \geq 2$ پس

(2) $\alpha - \beta + 1 = |vy| + 1 \geq 2$

طبق لم داریم $|vy| \geq 1$ است

طول رشته‌ای که حاصل شد یک عدد مرکب است ← در زبان نیست ← CFL نیست

مقدار n ها نامربع کامل است

$\boxed{ex8}$ $L=\{a^{n^2}, n\geq 0\}$ is not CFL .

فرض می‌کنیم CFL است ← یک p داریم که رشته‌های به طول حداقل p را

می‌شه pump کرد که $|s|\geq p$. اگر $(n=p^2)$

$s=a^{p^4}$ → $|s|\geq p$, $s\in L$ →

یک تجزیه‌ای مثل $s=uvxyz$ داریم که ۳ شرط لم را ارضا می‌کنه

یعنی ① $vy\neq\varepsilon$, ② $|vxy|\leq p$ ③ uv^ixy^iz

به ازای i جزو L باشه

طبق شرط دوم که $vy\neq\varepsilon$ ← $vy=a^k$ است که $\boxed{k\geq 1}$ است

اگر p pump کنیم و $\boxed{i=2}$ بگذاریم uv^2xy^2z ← s' ←

$\boxed{|s'|=p^4+k}$ $s'=a^{p^4+k}$ →

اما این رشته توی L قرار نمی‌گیره چون

یعنی طول رشته‌ی بعدی p^4+2p^2+1 ← $a^{(p^2+1)^2}$, $\cancel{a^{p^4}}$

$a^{(p^2)^2}$ اما طول s' هم $\boxed{p^4+k}$

$p^4<|s'|<(p^2+1)^2$

اگر k بخواهد فوری را به $2p^2+1$ برسانه ←

$\boxed{K=|vy|}$ است $|vxy|\leq p<2p^2+1$

و $K=|vy|$ → $|vy|\leq|vxy|\leq p<2p^2+1$

پس k ما اکیداً کمتر از $2p^2+1$ است ←

طول رشته‌ی uv^2xy^2z ما اکیداً از p^4 بیشتر و اکیداً کمتر از

$(p^2+1)^2$ است ← داخل زبان نیست ✗ ← CFL نیست

★ اگر یک زبان CFL باشد ← لم تزریق برایش برقرار است اما اگر

لم تزریق برقرار باشد ← الزاماً نمی‌شه گفت مستقل از متن است

برقراری لم تزریق CFL بودن را = نشان نمی‌دهد

برای اثبات CFL بودن ← یک گرامر / یک PDA

$L = \{a^k b^l c^m \mid m = kl\}$ (تمرین)

قرار است نشان دهیم L یک CFL نیست

الف) چرا رشته‌ی $s = ab^p c^p$ برای رسیدن به تناقض یک مثال نیست هستیم

چون p تعداد کمی از b و c وجود دارد را تغذیه می‌کنند

یعنی این رشته ما را به تناقض نمی‌رساند

ب) با بهره‌گیری از رشته‌ی $s = a^p b^p c^{p^2}$ نشان دهید که این زبان CFL نیست. راهنمایی: دو حالت را در نظر بگیرید ← ① vy شامل a باشد

② vy شامل a نباشد

② برای $|vy| = k_1$ تا b و k_2 تا c دارد

که $k_1 + k_2 \geq 1$

است. باید نشان دهیم وقتی رشته را pump up می‌کنیم توی زبان نمی‌افتد

$uv^2xy^2z = a^p b^{p+k} c^{p^2+k_2}$

$p^2 + k_2$ و $p(p+k_1)$

یعنی باید $\cancel{p^2} + k_2 = \cancel{p^2} + pk_1$ ← یعنی $\boxed{k_2 = pk_1}$

اگر k_1 صفر باشد ← k_2 هم صفر است و این تناقض دارد با $k_1 + k_2 \geq 1$ ← رشته داخل L نیست

برای اثبات CFL نبودن می‌توان در کنار لم تزریق از خواص بستاری هم استفاده کرد ← اشتراک یک زبان CFL و یک زبان regular یک $\boxed{CFL}$ است.

مثال: $L = \{w \in \{a,b,c\}^* \mid n_a(w) = n_b(w) = n_c(w)\}$ یک CFL نیست

$L \cap L(a^*b^*c^*)$ معادل با $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ است

regular

می‌دانیم که $L_1 = L \cap L(a^*b^*c^*)$

L نمی‌تواند CFL باشد، اگر بود L_1 هم باید CFL باشد که با اثبات تناقض دارد

regular ∩ CFL = CFL اشتراک

یا اینکه می‌دانیم

L_1 یک CFL نیست و قبلاً اثبات کردیم ← L هم CFL نیست

Session 17

FA ها مدل های خوبی هستند برای دستگاه هایی که حافظه ی محدودی دارند

PDA ها و NPDA ها هم مدل های هستند که تا اندازه ی حافظه ی نامحدودی داریم که اون حافظه به شکل stack است

یک سری زبان ها که CFL نیستند، خارج از قدرت PDA ها هستند ← باید با TM (Turing machine) نمایش بدیم.

یک Turing machine می تواند همه کارهای که یک کامپیوتر واقعی انجام می دهد را انجام دهد. یک TM شبیه یک FA است ولی یک حافظه نامحدود دارد

اگر اثبات کنیم یک تسک بالاتر از قدرت یک TA است ← اثبات کردیم هیچ کامپیوتری از انجام اون کار بر نمیاد

TM ها قادر هستند هر الگوریتمی را اجرا کنند.

تز تورینگ ادعا می کند که TM ها قوی ترین نوع automata هستند.

هدف اصلی تورینگ مشخص کردن محدودیت های ذاتی الگوریتم ها (محدودیت الگوریتم ها)

TM یک مدل انتزاعی است و قرار نیست implement شود

tape ← ماشین تورینگ مجموعه ای از تعدادی state است که یک نوار دارند ← الفبای نوار B

دو الفبا ← ① رشته ای که الفبای ورودی روی آن

② الفبای نوار

مثل PDA ها که دو الفبا داشتند (ورودی و استک)

نوار از تعدادی خانه مجزا تشکیل شده

طولش نامحدود است

از سمت چپ بسته شده ولی از راست نامحدود است

خانه هایی از نوار که خالی هستند را با blank symbol (⊔) نشون می دیم

① این tape و input device است یعنی ورودی ما روی این نوار قرار می گیرد

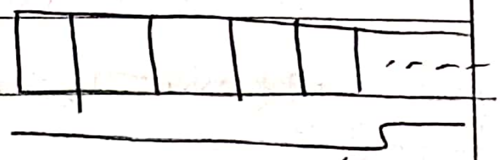
② حافظه ی ما هم هست یعنی در حین انجام محاسبات از این نوار هم استفاده می کنیم

③ اگر قرار باشد چیزی را به عنوان خروجی گزارش کنیم ← از این نوار به عنوان خروجی استفاده می‌شود

پس کاراکترهای tape ← ① ورودی

② در حین عملیات به شکل حافظه

③ در پایان عملیات به عنوان خروجی



شکل نوار

هر کدام از این خانه‌ها یک سمبل را در بر می‌گیرد

برخلاف DFA ها و NFA ها و PDA ها ← در TM ها تمام رشته ورودی را کاراکتر به کاراکتر نخوانیم تا وقتی تمام شود

TM ها به هر شکل دلخواهی می‌توانند ورودی را بخوانند. head ای که روی نوار

هست به هر شکل دلخواهی می‌تواند حرکت کند

توقف پس چه زمانی انجام می‌شود؟

* در ابتدای کار ورودی را در سمت چپ‌ترین خانه می‌گذاریم مثلاً رشته‌ی abaab

head

a	b	a	a	b	⊔	⊔	⊔	...

خالی خالی

TM ها لزومی ندارد که متوقف شوند یعنی یا ① accept می‌شود یا ② reject

③ loop (هیچ وقت تمام نمی‌شود)

یک TM هیچ وقت به بیش از ۲ تا state برای توقف (halt) نیاز ندارد

q_{accept} و q_{reject}

مثل DFA ها نیست که مجموعه‌ای از final ها داشته باشیم. اگر کار به state

accept یا reject ختم شد ← بلافاصله توقف انجام می‌شود

اگر یک TM هیچ وقت وارد accept یا reject نشود ← تا ابد به کارش ادامه می‌دهد

loop

زبان تشخیص ماشین: فقط رشته‌هایی که توسط ماشین accept می‌شوند

پس یک تورینگ ماشین مثل M روی یک رشته ورودی مثل w معین نیست که قبول

بشه یا رد (خاتمه پیدا کنه)

Scanned by CamScanner

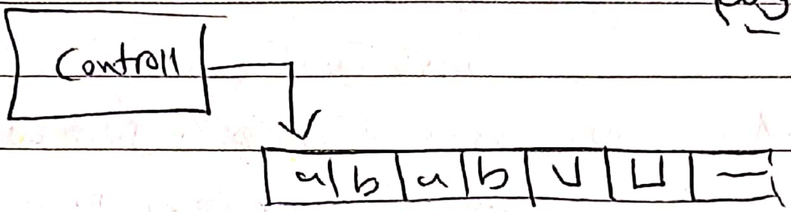
* تفاوت های TM ها و FA ها

1) یک TM می‌تواند هم روی tape بنویسد هم بخواند

2) read write head می‌تواند هم به سمت راست برود هم چپ

3) tape (نوار) نامحدود است

4) یعنی اینکه وارد یک state با accept یا reject شدیم ← متوقف می‌شیم



ex1) $B = \{w\#w \mid w \in \{0,1\}^*\}$.

یک TM مثل M_1 را برای زبان بالا بنویسید چون $w\#w$ یک CFL نیست ← یک CFG یا PDA براش وجود نداره.

اول head

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ ...

کار کاراکتر اول را خط می‌زند

اگر صفر بود خطش می‌زنه و x می‌گذارد

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔

head خودش را می‌رسونه به کاراکتر اول بعد از #

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ اگر 1 بود ← reject می‌کنه

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔

بعد دوباره میره اول کار

x x 1 0 0 0 # x 1 1 0 0 0 ⊔ سر کاراکتر دوم خطش می‌زنه و

x x x x x x x # x x x x x x x ⊔ جای x می‌گذارد

accept

head حرکت می‌کنه میره سر کاراکتر دومی که بعد از # است ← اگر 1 بود x می‌گذاره جاش اگر صفر بود reject و

اگر همه‌ی این کاراکترها با هم خط خوردن و کاراکتری باقی نمانده که خط زده نشده باشه ← accept می‌کنه

تابع انتقال

الگوریتم M_1 روی رشته ورودی w

① روی نوار و zigzag کن (در هر position ای که هستی نسبت به موقعیت متقابل و زیگزاگ کن) تا ببینی مثل هم هستند یا نه. اگه مثل هم نبودند یا هیچ # ای پیدا نشد ← رشته را reject کن

سمبل ها را خط بزن (جا بپوش، n بگذار) وقتی داری با سمبل متقابلش در آن سمت # چک می کنی

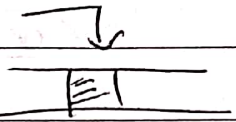
② وقتی همه سمبل های سمت چپ # خط زده شدند ← چک کن آیا سمبلی هست که در سمت راست # باشه و تا حالا خط نزده باقی مانده؟ اگر هست ← reject کن

اگر نیست رشته را قبول کن

در چه state ای هستم — سمبل زیر head چیه

$\delta(q, a)$ — ورودی تابع انتقال = یک زوج مرتب

$a \in \Gamma$



حرکت در TM: ① state عوض شود (مثلاً بریم به q' که یک state دیگر است)

یا ② سمبل زیر head عوض شود

③ head که می خواهد حرکت کند سمت چپ بره یا راست؟

در هر حرکت یک خونه فقط می تونه راست بریم یا چپ (یعنی از یک خانه بیشتر نمی تونه حرکت کنه)

زیر head را چی بذاریم — left — right

$$Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

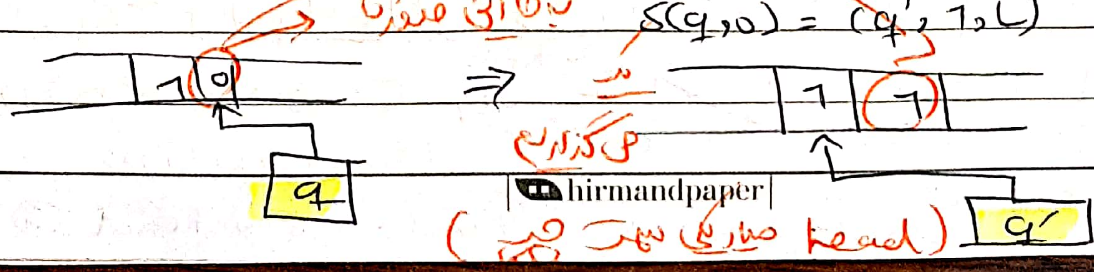
در چه state ای هستم — زیر head چیه — برگردیم به چه state برود — head را چپ ببریا راست

برو چپ

مثال $\delta(q, a) = (r, b, L)$

مثال ۲ $\delta(q, 0) = (q', 1, L)$

به جای صفر — یک — می گذاریم



(حرکت head به سمت چپ)

تعریف رسمی ماشین تورینگ ← ۷ تایی مرتب

$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

جایی که Q, Σ, Γ ، یک سری set های محدودی هستند :

① Q : set of states

② Σ : الفبای رشته ورودی که شامل $\boxed{\sqcup \text{ نیست}}$

③ Γ : $(\Sigma \subseteq \Gamma)$ و $\sqcup \in \Gamma$ که الفبای tape است

deterministic

④ $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: تابع انتقال

⑤ $q_0 \in Q$: start state

⑥ $q_{accept} \in Q$: accept state

⑦ $q_{reject} \in Q$: reject ~

$q_{reject} \neq q_{accept}$

$\Sigma \subseteq \Gamma \ , \ \sqcup \notin \Sigma \ , \ \sqcup \in \Gamma$

چون ⊔ توی Σ نیست ← اولین ⊔ ای که ببینم یعنی رشته ورودی تمام شده

اگر M بخواهد که head را از سمت چپ‌ترین موقعیت ببرد بازم به چپ head همچنان همان جا می‌ماند (leftmost)

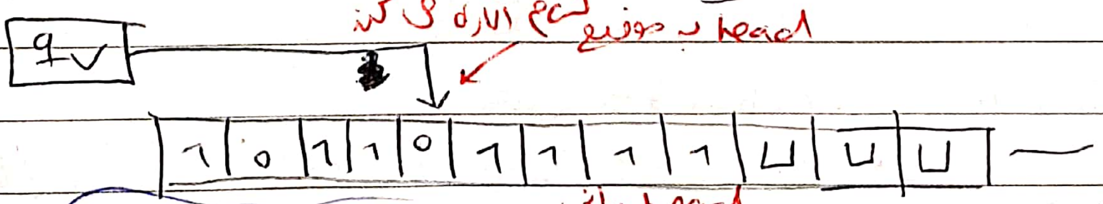
ماشین محاسبه ادامه پیدا می‌کند تا وقتی که وارد q_{accept} یا q_{reject} بشود ← توقف

در این صورت متوقف می‌شود در غیر این صورت M تا ابد به کار خودش ادامه می‌دهد

(Configuration of TM)

(نشان دادن وضعیت آنی تورینگ ماشین در حال حاضر)

head ← موقعیت رو نشان می‌کند



$1011 \, q_7 \, 01111$

بالای head اشاره می‌کنه

① در چه حالتی state هست

② محتوای نوار چی هست ؟

③ head به کجا اشاره می‌کنه ؟

محتوای نوار سمت چپ head

state

config ـ C_1 و config ـ C_2 رای دهد اگر بتوانیم به صورت مجاز از C_1 به C_2 برویم در یک step

$$\boxed{C_1 \vdash C_2}$$

انجام دهد C_2 را در یک گام

اگر کانفیگ فعلی ما

C_1 و $u a q_i b v$ $\qquad \delta(q_i, b) = (q_j, c, L)$

تابع انتقال این باشد

← C_2 چه خواهد بود

$C_1 \vdash C_2$ $\qquad C_2$ و $u q_j a c v$

به جای b و c بگذار

$u a q_i b v \vdash u q_j a c v$

u, v سمبل‌های کاراکتر هستند که $(u, v \in \Gamma^*)$ هستند.

state عوض شد ولی به چپ رفتیم

$a, b, c \in \Gamma$ هستند

اگر تابع انتقال باشد

به جای b بنویس c — برو راست

$\delta(q_i, b) = (q_j, c, R)$

$u a q_i b v \vdash_M u a c q_j v$

(از state q_i برو به q_j)

نماد $\vdash^*$ یعنی تعدادی گام صفر یا بیشتر از کانفیگی به کانفیگ دیگر می‌رسیم به راحتی

$C_1 \vdash^* C_2$ یعنی تعدادی گام (صفر یا بیشتر) از C_1 رسیدیم به C_2

$\vdash_M$ یعنی حرکت از کانفیگ ۱ به ۲ در ماشین M الف

ترتیب head به سمت چپ

① در state q_0 ② هستیم در ابتدای کار ③ روی نوار الف و m موضع

$$L(M) = \{ w \in \Sigma^* \mid q_0 w \vdash_M^* \alpha\, q_{accept}\, \beta \text{ for some } \alpha, \beta \in \Gamma^* \}$$

از کانفیگ ابتدای طی تعدادی گام برسیم به کانفیگ

در (رشته) اشاره می‌کند

$\alpha\, q_{accept}\, \beta$

قبل از head

بعد از head هرچیزی هست.

$\boxed{94}$ β $\boxed{q_i bv}$ وضعیت فعلی

$\boxed{b}$

اگر $\delta(q_i, b) =$

(q_j, c, L)

چون head را فقط یک خانه از اول چپ‌تر ببریم ← head روی B می‌ماند

$q_i bv \vdash q_j cv$

اگر $\delta(q_i, b) = (q_j, c, R)$

$q_i bv \vdash c q_j v$

کانفیگ uaq_i برابر کانفیگ $uaq_i \sqcup$

★ کانفیگ آغازی M روی رشته w ← $\boxed{q_0 w}$

① روی state آغازی هستیم

② w روی tape قرار داره ③ head داره به سمت چپ‌ترین موقعیت اشاره می‌کنه

وقتی کانفیگ‌ها به شکل $\alpha q_{accept} \beta$: accept configuration

halting configuration

و $\alpha' q_{reject} \beta'$: reject configuration

به معنی اینکه به accept یا reject برسیم ← توقف رخ می‌دهد و مهم نیست محتوای نوار چی هست و head به کجا اشاره می‌کند

تابع انتقال را می‌شه بین stateها q_{accept}، q_{reject} تعریف کرد به طوری که:

$\delta : Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

چون از q_{accept} و q_{reject} هیچ وقت خروجی نداریم و تابع انتقالی از اون‌ها نداریم ← از Q می‌شه حذفشون کرد برای تابع انتقال ← دامنه δ : $\boxed{Q' \times \Gamma}$

سوال‌ها

① آیا TM می‌تواند ⊔ را روی tape بنویسد؟

② آیا الفبای نوار (Γ) می‌تواند برابر با الفبای ورودی (Σ) باشد؟

③ آیا head یک TM می‌تواند در دو گام متوالی جای ثابتی (توقف) نکند و یک جا بماند؟

آره می‌شه. مثلاً وقتی که در leftmost هستیم و بخواهیم یک مکان به چپ بریم ← سر جای ثابت می‌ماند

④ آیا یک TM می‌تواند یک state واحد داشته باشد؟

(★ تعریف رسمی مفهوم محاسبه)

یک TM به نام M یک رشته w را قبول می‌کند اگر یک سری کانفیگ‌های مثل $C_1, C_2, \dots, C_k$ داشته باشیم که

① C_1 کانفیگ start باشد $\boxed{q_0 w}$

② هر C_i، C_{i+1} را بدهد $C_p \vdash C_{p+1}$

③ C_k یک accepting config باشد $\alpha\, q_{accept}\, \beta$

$C_1 \vdash C_2 \vdash \dots \vdash C_m$

(عبارات مهم در نظریه محاسبه پذیری)

① یک TM رشته w را accept می‌کند اگر با رشته w وارد q_{accept} شود

② یک TM به نام M رشته ورودی w را reject می‌کند اگر به ازای w وارد q_{reject} شود.

③ یک TM به نام M روی w متوقف می‌شود اگر ... یا ... وارد یکی از دو state q_{accept} یا q_{reject} شود

④ TM روی رشته ورودی w به loop می‌خورد اگر ... یا ... ماشین نتواند وارد q_{accept} یا q_{reject} شود. یعنی روی w (توقف ندارد)

(★ Turing recognizable) ← یک زبانی را turing recognizable می‌گیم اگر یک ماشین تورینگ داشته باشیم که این زبان را recognize کند ← اگر یک TM یک زبان را accept کند

به اون زبان Turing recognizable همه اینها دقیقاً رشته‌هایی که رد می‌شوند

رشته‌هایی که داخل زبان L است ← TM وارد q_{accept} می‌شه

نیستند ← TM یا وارد q_{reject} یا loop می‌شود

وارد q_{reject} نشده اند و ممکن است چون توی loop هستن ها رد نشده باشن

Turing acceptable (1) ← یعنی Turing recognizable

semidecidable (2)

recursively enumerable $\boxed{RE}$

Deciding & decider

یک TM ممکن است روی بعضی رشته‌ها لوپ بخوره و halt نشه اما یک TM ای که روی همه‌ی ورودی‌ها همیشه متوقف می‌شه (reject یا accept) ← بهش decider می‌گن

یعنی روی هیچ ورودی‌ای لوپ نمی‌خوره ★ ما همواره decider ها را ترجیح می‌دهیم.

★ برای یک TM ای که decider نیست ما نمی‌دانیم که اگر L [illegible] طول کشیده داره جوابو انجام می‌دهد یا توی لوپ افتاده ولی وقتی یک decider داریم مطمئن هستیم متوقف می‌شه. ★ هیچ وقت loop نمی‌خوره

(decider ای که یک زبان را recognize کند می‌گیم اون زبان را decide کرده)

زبان Turing decidable است اگر زبانی توسط یک TM ای decide بشود

اگر یک TM ای که decider است یک زبان رو accept کنه ← به اون زبان recursive یا decidable می‌گن

یک TM ای به نام M که یک زبان مثل L را decide می‌کنه اگر و فقط اگر برای هر رشته w ای که $w \in \Sigma^*$ است M وارد q_{accept} بشود برای رشته‌هایی که داخل L هستند و وارد q_{reject} بشود برای رشته‌هایی که توی زبان L نیستند.

★ اگر یک TM ای L را decide کنه ← L را accept می‌کنه

★ اگر L را accept کنه ← ممکن است که L را decide نکنه

deciding یک نوع قوی‌تر است نسبت به accepting

مرز بین decider ای recognizable Turing هم هست اما
مرز مشخص نیست

accepting ← رشته های داخل زبان را بپذیرد و وارد q_{accept} شود
q_{reject}
نیستند ← ...
loop

deciding ← رشته داخل زبان ← q_{accept}
نیست ← q_{reject}
loop ندارد

transformer ← یه TM ای که وقتی halt می کند یک چیزی را به عنوان output روی نوار می نویسد
مثل این است که یک تابع را دارد روی ورودی run می کنند و رشته ای را هم می خواهند به عنوان خروجی به کاربر برگردانند

* یک TM به نام M یک function به نام f را محاسبه می کند
$f: \Sigma^* \mapsto \Sigma^*$
اگر یک رشته w به عنوان ورودی به M بدهیم که کانفیگ را روی start config ($q_0 w$) آغاز کند و در انجام halt کند با یک رشته $\boxed{f(w)}$ روی $\boxed{\text{tape}}$

session 18

$\boxed{\text{ex1}}$ یک TM ای به نام M داریم که زبان $A = \{0^{2^n} \mid n \geq 0\}$ را decide می کند
حاوی رشته های 0 است که طولشان $\boxed{\text{توانی از 2}}$ است

الگوریتم M (M روی رشته ورودی w)

① یکی درمیان از چپ شروع می‌کنیم صفرها را خط می‌زنیم (x می‌گذاریم)

(توصیف کلی از روند کار)

یک بار از اول تا آخر می‌ره ← مقدار صفرها نصف میشه

(اگر ۱۶ تا صفر داریم اول کار ← میشه ۸ تا) ←

② اگر اول کار یک صفر باقی بماند ← رشته accept میشه

③ اما اگر اول کار یک تعداد فردی صفر (بیشتر از یک) داشته باشیم رشته را reject می‌کنیم

④ اگر تعداد مقدار صفرها زوج است ← head میره روی leftmost در tape و بعد مرحله ۱ دوباره تکرار میشه ⑤

مثال e ①

20 → 10 → ⑤ رشته reject میشه ④

یک تعداد فردی صفر داریم که ≠ 1 است

x0x0x0x0x0
x x x x

ø ⊗ 0 ⊗ ⊗
⊔

② 72 → 36 → 18 → 9 ≠ 1 (عدد فرد)

③ 32 → 16 → 8 → 4 → 2 → 1

فرض نصف کنیم ما را به عدد ۱ رسونده ← 32 توانی از ۲ بوده

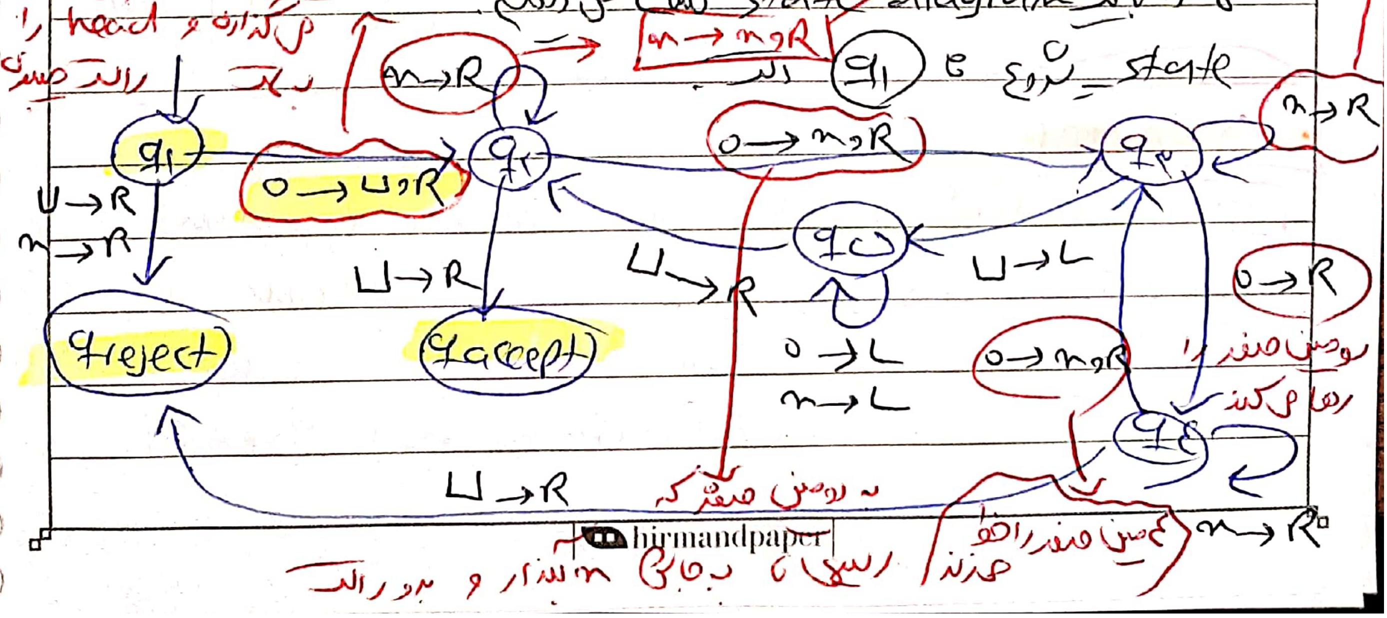
ø ø 0 ø 0 ⊔   ø x ø x 0 x ⊔ — xها را رد می‌کند (skip) می‌کند (توصیف کلی)

$M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$

$Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$

$\Sigma = \{0\}$ $\Gamma = \{0, x, \sqcup\}$ بجای صفرهای فرد x بذار و برو به راست

δ را با یک state diagram نشان می‌دهیم q_1 state شروع



$(0 \to \sqcup, R)$

* چرا اول کار میاد $\sqcup$ را می‌ریزه توی tape ؟

اولین صفر را $\sqcup$ می‌گذاریم تا حواسمون باشه سمت چپ‌ترین خونه کجاست

مثل علامت \$ توی stack

(برای وقتی که نصف می‌خواهیم کنیم) دوباره

به جای 0 ها x بگذار و به سمت راست برود

$(x \to x, R)$

یعنی $x \to R$

$\cancel{0}\,\cancel{0}\,0\,\cancel{0}\,0\,\cancel{0}\,0\,\cancel{0}\,\sqcup$

$\sqcup$

مرحله بعدی

$\sqcup\ x\,\cancel{0}\,x\,\cancel{0}\,x\,\cancel{0}\,x\,\sqcup$

اگر توی q_4 بودی و به آخر رشته رسیدی یعنی

یعنی فرد تا صفر دیدی و باید reject کنی →

$\sqcup \to R$ از سمت راست که $\sqcup$ آخر رشته را دیدی

تعداد صفرهایی که ما داریم اگر فرد باشد و غیر از یک باشد

وقتی توی q_4 هستیم ← فرد تا صفر دیدیم

q_{reject} ← q_4

$\sqcup \to R$

اگر توی q_1 برگشتیم و اینکه رشته را دیدیم ← فرایند نصف کردن باید تکرار شود

زوج تا صفر داریم

یعنی x ها و 0 هایی که دیدی را رد کن و head را ببری سمت چپ

برو چپ تا برسی به $\sqcup$ اولین رشته در سمت چپ‌ترین تا به ابتدای رشته برسی

وقتی این بلنک ($\sqcup$) را دیدی یکی جلو برو و دوباره از ابتدای رشته شروع

به بررسی می‌کنیم ← وقتی همه x ها را رد کردی به بلنک ($\sqcup$) رسیدی

q_{accept}

رفتن به q_{accept} یعنی همه x ها را رد کنیم و

به یک $\sqcup$ برسیم در آخر رشته

trick این است که صفر اول را ($\sqcup$) بگذاریم تا نشانه‌ای باشد که سمت چپ‌ترین

موقعیت رشته کجاست *

ابتدای state — head را ابتدای نوار

$q_1 0000 \vdash \sqcup q_2 000 \vdash \sqcup x q_3 00 \vdash \sqcup x 0 q_4 0$

صفر ها رو صفر رو هم

بجای اولین صفر x میذاریم / یک صفر در میان x میذاریم

سمت چپ ترین حالت

اول tape

$\vdash \sqcup x 0 x q_3 \sqcup \vdash \sqcup x 0 q_5 x \sqcup \sqcup \vdash$

صفر های جا خورده رو

$\sqcup x q_5 0 x \sqcup \vdash \sqcup q_5 x 0 x \sqcup \vdash q_5 \sqcup x 0 x \sqcup$

$\vdash \sqcup q_2 x 0 x \sqcup \vdash \sqcup x q_2 0 x \sqcup \vdash$

$\sqcup x x q_3 x \sqcup \vdash \sqcup x x x q_3 \sqcup \vdash \sqcup x x q_5 x \sqcup$

$\vdash \sqcup x q_5 x x \sqcup \vdash \sqcup q_5 x x x \sqcup \vdash q_5 \sqcup x x x \sqcup$

$\vdash q_5 \sqcup x x x \sqcup \vdash \sqcup q_2 x x x \sqcup \vdash \sqcup x q_2 x x \sqcup \vdash$

$\sqcup x x q_2 x \sqcup \vdash \sqcup x x x q_2 \sqcup \vdash \sqcup x x x \sqcup q_{accept}$

قبول شد

$\sqcup x x x \sqcup q_{accept}$

آخرین Config

$q_1 00000 \vdash \sqcup q_2 0000 \vdash \sqcup x q_3 000 \vdash$

$\sqcup x 0 q_4 00 \vdash \sqcup x 0 x q_3 0 \vdash \sqcup x 0 x 0 q_4 \sqcup$

$\vdash$

$\sqcup x 0 x 0 \sqcup q_{reject}$

ردش کنه

$\boxed{ex2}$ $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$

$B = \{w\#w \mid w \in \{0,1\}^*\}$.

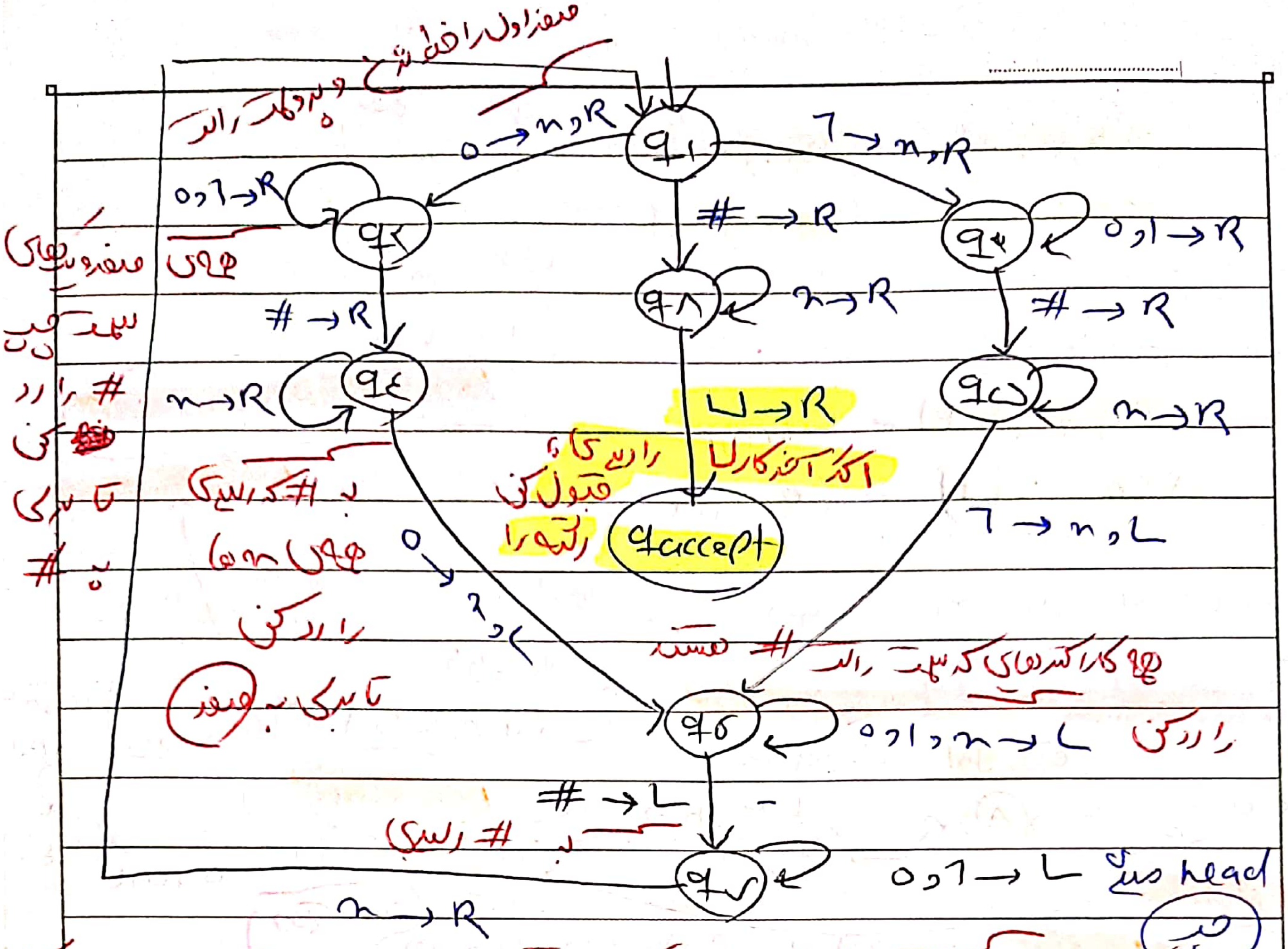
$Q = \{q_1, q_2, \ldots, q_8, q_{accept}, q_{reject}\}$

$\underbrace{\Sigma = \{0,1,\#\}}_{\text{الفبای ورودی}}$, $\underbrace{\Gamma = \{0,1,\#,x,\sqcup\}}_{\text{الفبای نوار}}$

الفبای ورودی — الفبای نوار

state diagram را نشان دهید

q_1 شروع state



همه ی کاراکترهایی که سمت چپ # هستند را دوباره x کن

تا برسی به آخرین چیزی که خط زدی

$\delta(q_3, 1) = (q_{reject}$, فرض ندارد به جای ۱ چی بگذاریم , فرض ندارد ببریم راست یا چپ $)$

اگر صفر خط زدی و به ۱ برسی سمت # راست باید reject کنی

$(q_{reject}, 1, R)$ مثلاً

سمت راست یا کدام هم صفر مثبت اگر جای یک رسی

باید اول خط # هم معادل ۱ باشد اگر صفر باشد ← reject کن

$\delta(q_5, 0) = (q_{reject}$, فرض ندارد جای صفر چی بگذاریم , فرض ندارد به کدام جهت ببریم $)$

برای accept کردن ← باید ببینه اگر آخر کار همه کاراکترها وجود ندارد

موقع accept کن ← جاهایی که سمت چپ # و اونهایی که x گذاشتی را x کنی

سمت راستش اونهایی که x هستند را x کنی اگر به ␣ رسیدی accept کن

0 , R

$\delta(q_8, 0) = (q_{reject}, \_ , \_)$ هیچی هیچی

$\delta(q_8, 1) = (q_{reject}, \_ , \_)$ هیچی هیچی

1 , L

$\delta(q_4, \sqcup) = (q_{reject}, \_ , \_)$ حداقل باید a ها بعد از b بخونیم

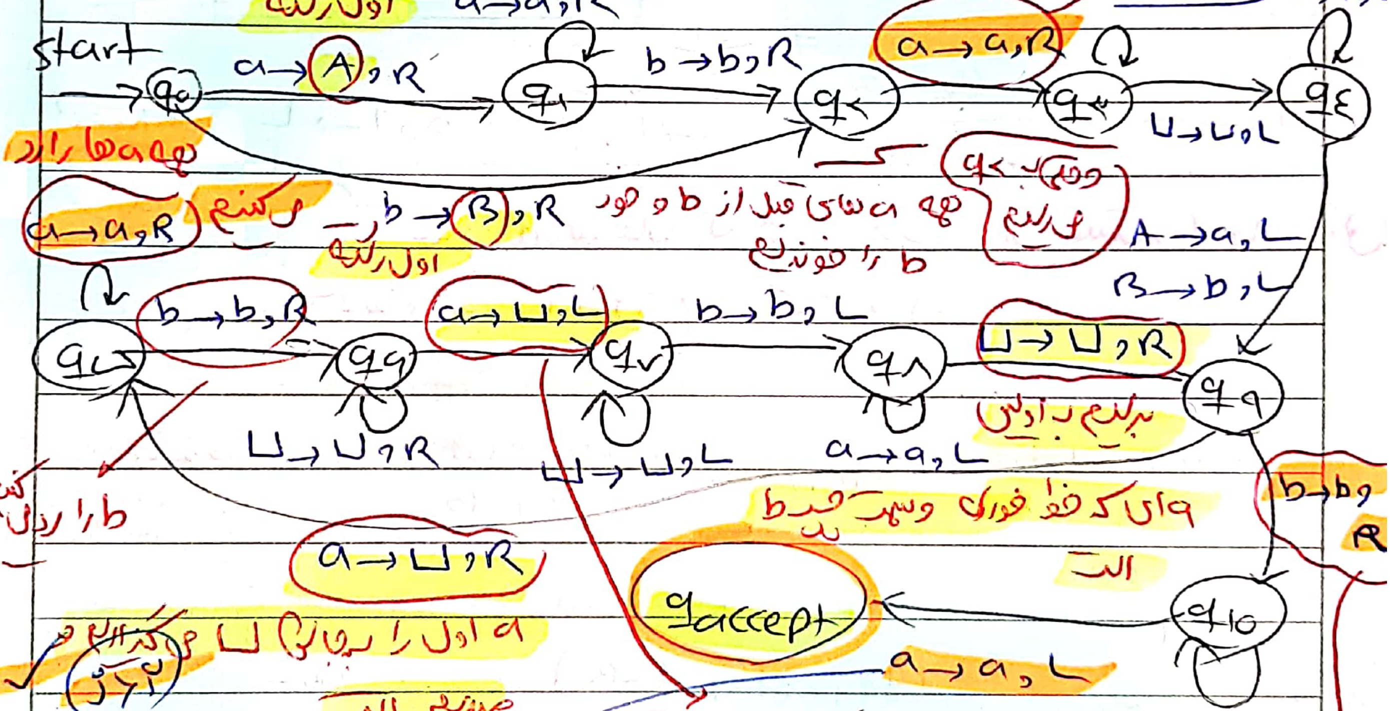
$\delta(q_5, \sqcup) = (q_{reject}, \_ , \_)$ چون $j > i$

یعنی سمت راست a ها کوتاه تر از سمت چپ است ← اگر توی q_5 ⊔ را دیدیم

EX $L = \{a^i b a^j \mid 0 < i < j\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, \sqcup, A, B\}$



a اول را بجای ⊔ می گذاریم ($i<j$)

می رسیم رشته

برمیگردیم به a ای که بعد از b است و خط نخورده

اگر a ای ببینیم که هنوز خط نخورده

همه a های که خط خورده را رد می کنیم

چون ۲ حرف داریم منفرد است ← اولین کاری که می کنیم می خونیم یا a است یا b

سعی می کنیم مطمئن شویم که فرم رشته درست است ← q_3 تا

$\boxed{a^* b a a^*}$ یعنی رشته به فرم

یعنی همه a های سمت راست b را خط بزنیم

یعنی از فرم مطمئن شدیم میایم تو q_6 دوباره رشته را چک می کنیم

سعی می کنیم برگردیم اول رشته و یک a از سمت چپ b خط می زنیم و یک a از سمت راست b و میریم ابتدای رشته که با A و B مشخص شده بود

بجای A ، a می گذاریم و میریم $A \to a, L$

به چپ ، head برنمی گردد

برای این رشته‌هایی که فرم را ندارند ولی $n \geq 2$ الف ← میفتیم توی loop

مثلاً رشته‌ی aba ← نباید accept شه یا reject شه یا بره توی loop

وقتی تعداد a‌های سمت راست و چپ b برابر باشه ← بعد از q_{10} بریم توی یک accept ای ← یک loop ی نهایی میشه

۲) اگر تعداد a‌هایی که سمت چپ b هستند بیشتر باشه از تعداد a‌های سمت راست

~~aa~~ aaab ~~a~~

⊔ ⊔

اینجا باید سمت راست b که چپ a ای دومین a سمت چپ را داشته باشیم

دومین a سمت چپ

دنبال a ای می‌گردیم که متناظر با a سمت چپ باشه

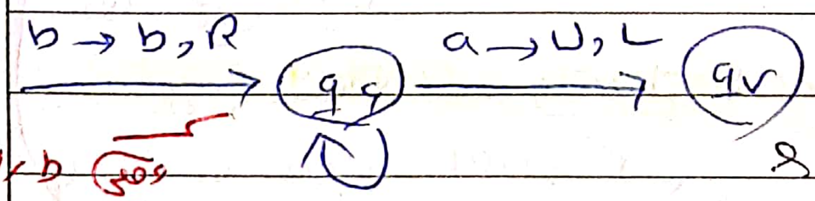
a ای نداریم ✓ → توی loop هامون افتادیم

الان توی این مثال q reject هامون کجا میشن و اگر این حالت‌ها loop هامون

اینجا این ماشین ما decider نیست چون امکان توی لوپ افتادن وجود داره

این یک recognizer است برای زبان L

(چون اینجا q reject نداریم الگوریتم نمی‌گیریم ...)



وقتی b را رد کنه

سوال الان q reject مون کجاست؟

میره دنبال a هایی که قبلاً خط زده و ردشون کنه

چون a ای دیگه نداریم

یعنی از اینجا ⊔ → a رد

گونه‌های ماشین تورینگ

session 19 — variants of turing machines

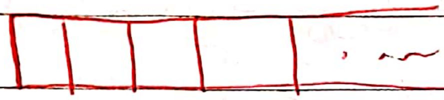
تغییرات باعث نمی‌شود که قدرت TM ، از حیث توصیف افزایش پیدا کند

در TM یک درجه‌ای robustness زیاد است

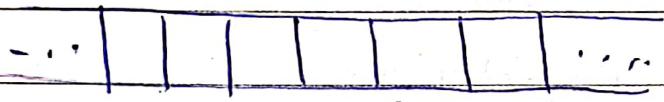
طرز تورینگ — می‌گه هر ماشینی که داشته باشید را می‌شود توسط یک Turing machine استاندارد، simulate کرد.

کارهایی وجود ندارد که بالاتر از قدرت یک TM استاندارد باشد

اولین تغییر در TM ، tape در به جای اینکه از یک سمت محدود باشد ، از دو طرف نامحدود باشد



tape 1 یک طرف نامتناهی — unilaterally infinite



tape 2 دو طرف نامتناهی — bilaterally infinite

قدرت TM هایی که روی tape دو طرفه نامتناهی است = قدرت TM های که یک طرفه نامتناهی هستند (ثابت)

می‌توان TM ای درست کرد که یک طرفه است — سمبل زیر head را می‌تواند e ... یک کار هم را می‌کند.

تغییر دوم multidimensional Turing machine

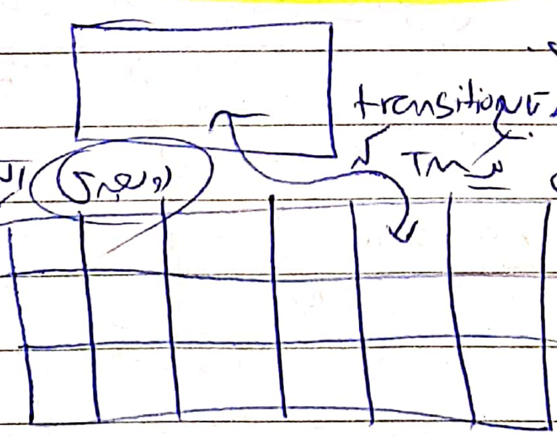
tape را ۲ بعدی یا ۳ بعدی کنیم

transition تابع $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times$ {L, R, U, D}

TM که سمبل زیر درجه state هست head

جای head را عوض کن

تغییر دهیم کامپیوتر



دو بعدی است — ۴ طرف — head می‌برد

down , up , left , right

می‌توان اثبات کرد قدرت TM هایی که روی tape چند بعدی است معادل TM های استاندارد است. ابعاد برای tape چیزی را به قدرت اضافه نمی‌کند.

تعریف ۶: TM with a stay option

head بعد از اینکه محتوای نوار را عوض کرد در جای فعلی بماند.

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

تابع انتقال TM استاندارد

تابع انتقال جدید:

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$

S: stay

theorem ۶: قدرت TM های که stay option را دارند معادل با TM های استاندارد است.

اثبات ۶: یک TM ای مثل M را باید simulate کنیم با یک TM ای مثل $\hat{M}$ که استاندارد است.

$M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$

$\hat{M} = (\hat{Q}, \Sigma, \Gamma, \hat{\delta}, \hat{q}_1, q_{accept}, q_{reject})$

اگر بتوانیم نشان دهیم $\hat{M}$ می‌تواند کار M را انجام دهد ← اثبات کرده‌ایم.

★ اگر $\delta(q_i, a) = (q_j, b, L \text{ or } R)$

$q_i \in Q$، $a \in \Gamma$

head به چپ یا راست حرکت کند ← $\hat{\delta}$ هم همین کار را کند

$\hat{\delta}(\hat{q}_i, a) = (\hat{q}_j, b, L \text{ or } R)$

تکثیر q_i توی Q ← $\hat{q}_i$ توی $\hat{Q}$

اگر $\delta(q_i, a) = (q_j, b, S)$

head در جای فعلی بماند ← هر کاراکتری می‌تواند باشد ($\in \Gamma$)

$\hat{\delta}$ ← تابع انتقال را طوری تعریف کنیم که head را یک بار به سمت راست ببرد بعد دوباره head را برگرداند به چپ ← یعنی در جای خود بماند

$\hat{\delta}(\hat{q}_i, a) = (\hat{q}_{j_S}, b, R)$ و $\hat{\delta}(\hat{q}_{j_S}, c) = (\hat{q}_j, c, L)$

$\hat{q}_{j_S}$

$\hat{q}_j$ یک state جدید تعریف

$\hat{q}_i$ $\hat{q}_j$

به جای مستقیم رفتن از $\hat{q}_i$ به $\hat{q}_j$ ← از $\hat{q}_i$ برود به $\hat{q}_{j_S}$ و از آنجا برود به $\hat{q}_j$

هر کاراکتری که دیدی از $\hat{q}_{j_S}$ برو به $\hat{q}_j$ و head را ببری یه چپ

$M \longrightarrow \hat{M}$

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{R, L, S\}$ $\hat{\delta}: \hat{Q} \times \Gamma \to \hat{Q} \times \Gamma \times \{L, R\}$

TM که stay-option دارد   TM استاندارد

توانستیم تغییر δ به $\hat{\delta}$ را ارائه کنیم ← یک TM معادل و استاندارد بسازیم

گرفتیم که TM اولی را simulate می‌کند ← این تغییر چیزی را به قدرت TM اضافه نمی‌کند

تعمیم ۴: multitape turing machine

$\delta: Q \times \Gamma^K \to Q \times \Gamma^K \times \{L, R, S\}^K$

زیر هر کدوم از head ها میره   زیر هر head چی   هر کدوم از head ها

اگر K تا tape داشته باشیم → K تا head خواهیم داشت   به جای سمبل حرکت بده

K تا سمبل زیر head بگذار

$\delta(q_i, a_1, \ldots, a_K) = (q_j, b_1, \ldots, b_K, L, R, \ldots, L)$

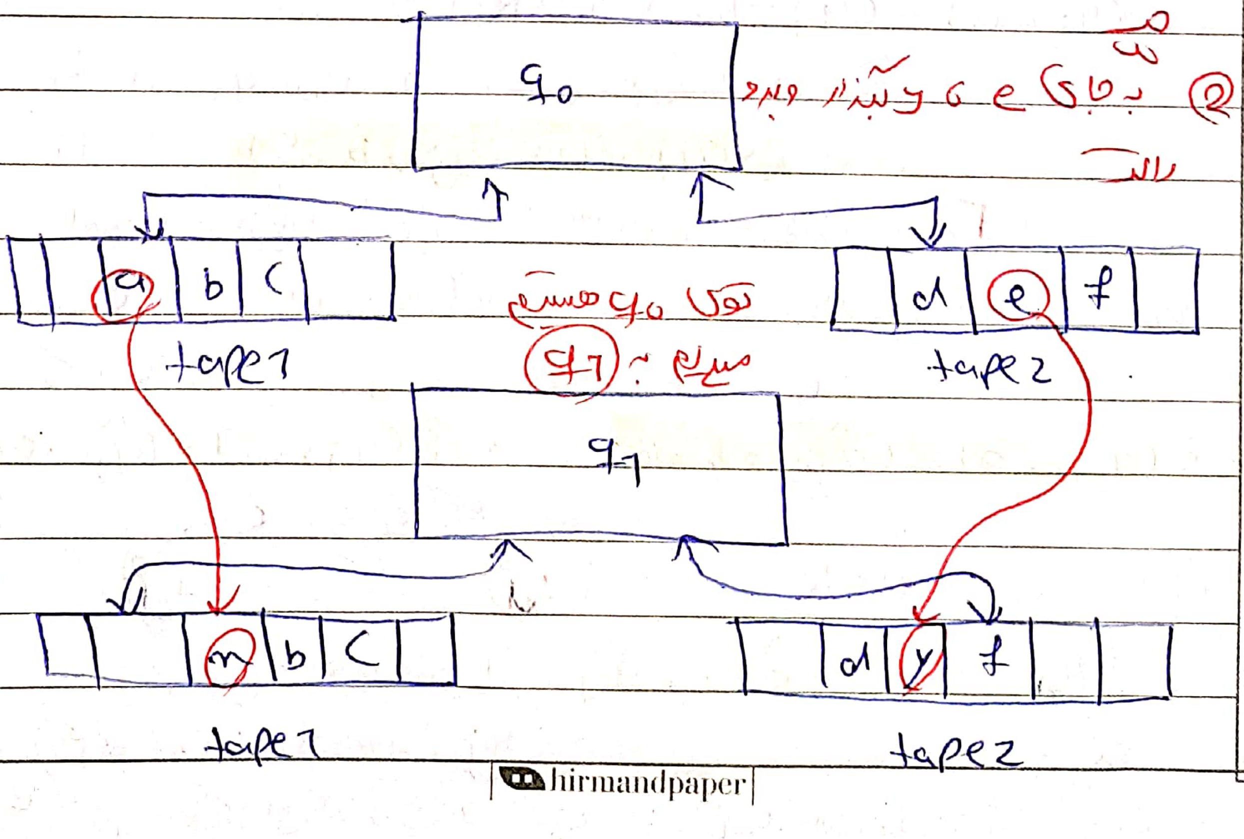
K تا سمبل که از زیر head است   K تا head   K تا جهت

مثال از یک multitape TM

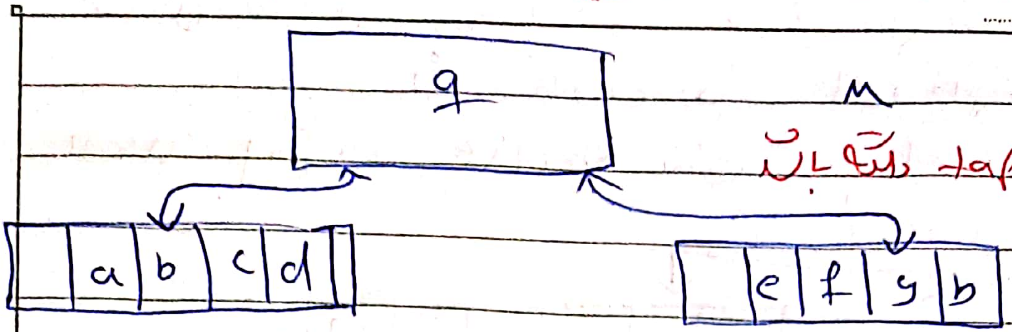
$\delta(q_0, a, e) = (q_1, x, y, L, R)$

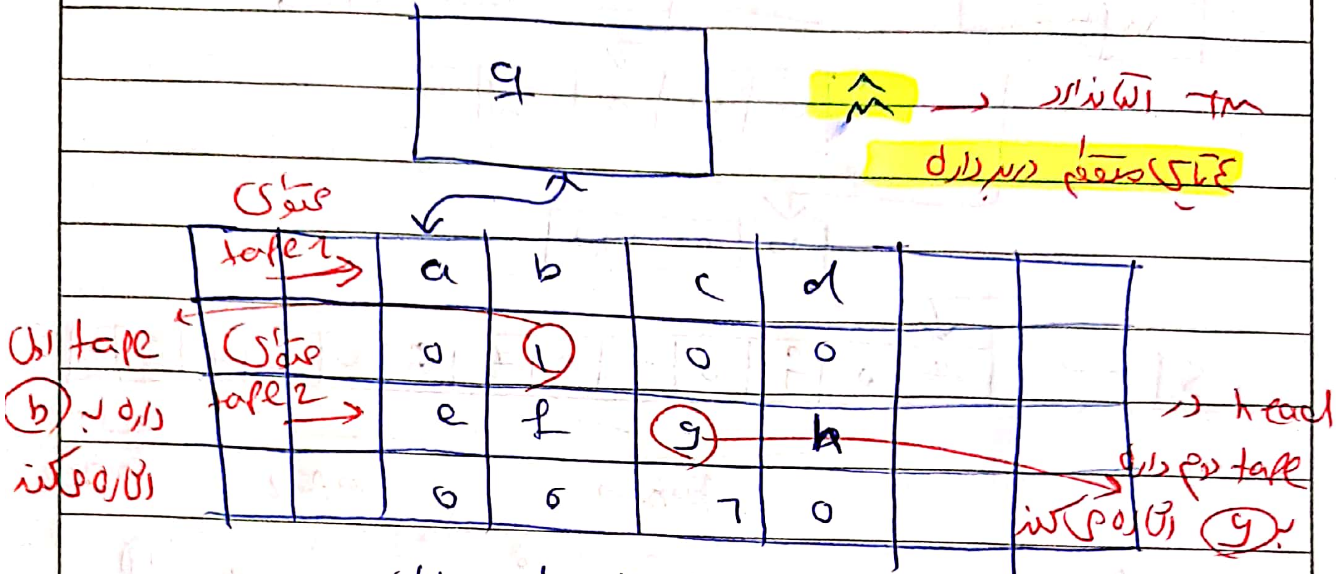
1) به جای a، x بگذار و برو چپ

2) به جای e، y بگذار و برو راست



M را با $\hat{M}$ simulate می‌کنیم





محتوای tape اول ← در track اول

دوم ← ، ، ، سوم

برای مشخص کردن موقعیت head ← از یک دنباله باینری الفبای جدید استفاده می‌کنیم

از عددهای یک علامت دهیم ۰

موقعیت head را نشان می‌دهد

در واقع از TM آلانداره بالا را می‌توان یک کاراکتر مرکب نشان داد

(d, 0, h, 0)

هر کدام از cell ها یک کاراکتر مرکب را در بر دارند ← سمبل های الفبای جدید ما هستند

کاراکتر مرکب هستند.

اثبات دوم برای multi tape ها: برای تبدیل کردن یک multitape TM

به الگوریتم M به یک single tape TM S

M که k تا tape دارد ← S معیار تا k تا tape را شبیه سازی

می‌کند با ذخیره کردن اطلاعات tape ها روی single tape

محتوای tape های مختلف را با tape ها را از هم جدا کنیم از # استفاده می‌کنیم

| # | | | | # | — | # | — | # |

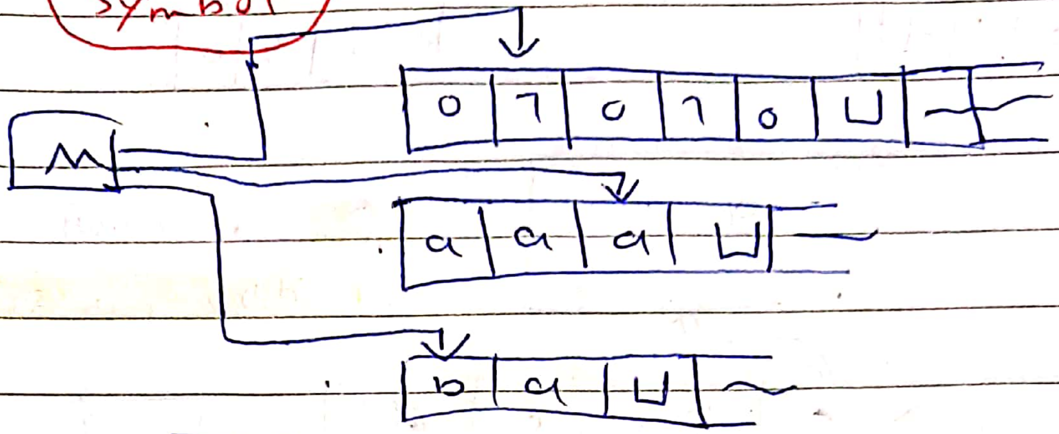
موضع هر head را چطوری نشان دهیم ؟

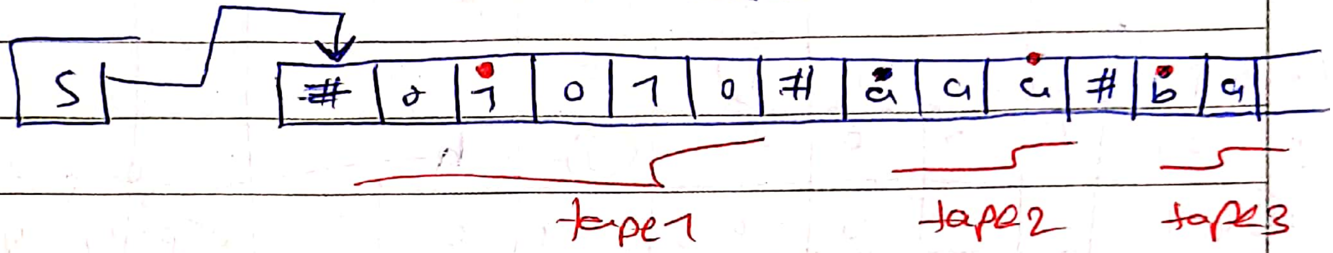
به ازای هر سمبلی که توی Γ هست ← یک سمبل نقطه‌دار رو اضافه می‌کنیم

$\Gamma = \{a, b, \sqcup\} \rightarrow \dot{a}, \dot{b}, \dot{\sqcup}$

دوتایی که [ȯ] دارند یعنی head روشون است

dotted symbol





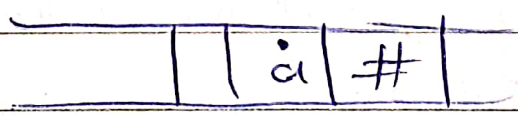
① $\# w_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$

محتوای نوار در اول کار

برای (S) input روی tape اول است و head هم به سمت چپ‌ترین کاراکتر اشاره می‌کنه و بقیه head ها جاهای خالی هستند

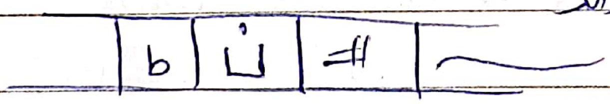
② باید (k+1) # بگذاریم برای جدا کردن محتوای نوارها

③ اگر



دیگه جا نیست

head روی a است و باید (بره) به سمت راست



کل محتوای که از # به بعد داریم یک واحد به راست شیفت می‌خوره

اگر جایی عبور شد head را ببری سمت راست و بیاد روی یه # ← کار 6

نقطه‌دار نمی‌شه بلکه # رو کلی به سمت راست حرکت می‌کنه تا ⊔̇ خواهیم داشت

و همه مقادیر بعدی

session 20

NTM — DTM

افزودن عدم nondeterminism قدرت TM ها را زیاد می کند

قدرت TM های استاندارد deterministic ≡ قدرت nondeterministic TM

مثل NFA ها و DFA ها ≠ مثل PDA ها و NPDA ها

افزودن عدم قطعیت باعث نمی شود قدرتش بیشتر شود

powerset

$\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$

مجموعه ای از option ها

$\delta(q, x) = \{(q', y, R), (q'', z, L)\}$

یک nondeterministic TM زمانی w را accept می کند که دنباله ای از حرکت ها باشد به شکل

$q_0 w \vdash^* x_1 \, q_{accept} \, x_2$

کانفیگ start w — در تعداد گام — accepting config

$x_1, x_2 \in \Gamma^*$ هستند — head به سمت چپ ترین سمبل w در اشاره می کند

$\delta(q, a) = \{(q_1, a_1, D_1), \ldots, (q_m, a_m, D_m)\}$

$\in Q$, $\in \Gamma$

TM option داریم

دنبال های حرکت ← به loop برسیم
به reject برسیم
به accept برسیم ✓ مهم این است

ex: $\delta(q_0, a) = \{(q_1, b, R), (q_2, c, L)\}$

$q_0 aaa \vdash b q_1 aa$ یا $q_0 aaa \vdash q_2 caa$

head ابتدا چون موقعیت سمت چپ ترین بوده و همچنین

برای NTM ها هم یک درخت محاسبه داریم — computation tree

کلاس زبان های CFL

NFA, NPDA, NTM → درخت های ندارند

DFA, DPDA, DTM

CFG ها

برای اثبات ← باید یک DTM به اسم (D) را بسازیم که همان NTM (N)

را simulate می‌کند ← D باید درخت محاسباتی N را تشکیل دهد

و توی درخت بگردد یک accepting config پیدا کند اگر حداقل یکی از این کانفیگ‌های

accepting پیدا کرد ← رشته را قبول می‌کند در غیر این صورت reject می‌کند

اگر درخت محاسبه متناهی باشد ← N همه‌ی branch ها توی reject ✓

رشته را halt کرده ولی اگر درخت نامتناهی باشد ← توی بعضی از

branch های (N) توی loop گیر کرده

اگر بیاییم عمق اول بریم ← اگر branch تا بی‌نهایت پیش بره و

درحالی که یه branch دیگه توی عمق ۵ رشته را accept کرده ← × منطقی نیست

$\boxed{\text{نقطه اول}}$ ← breadth_first search

اول نقطه اول را می‌بینیم بعد نقطه دوم

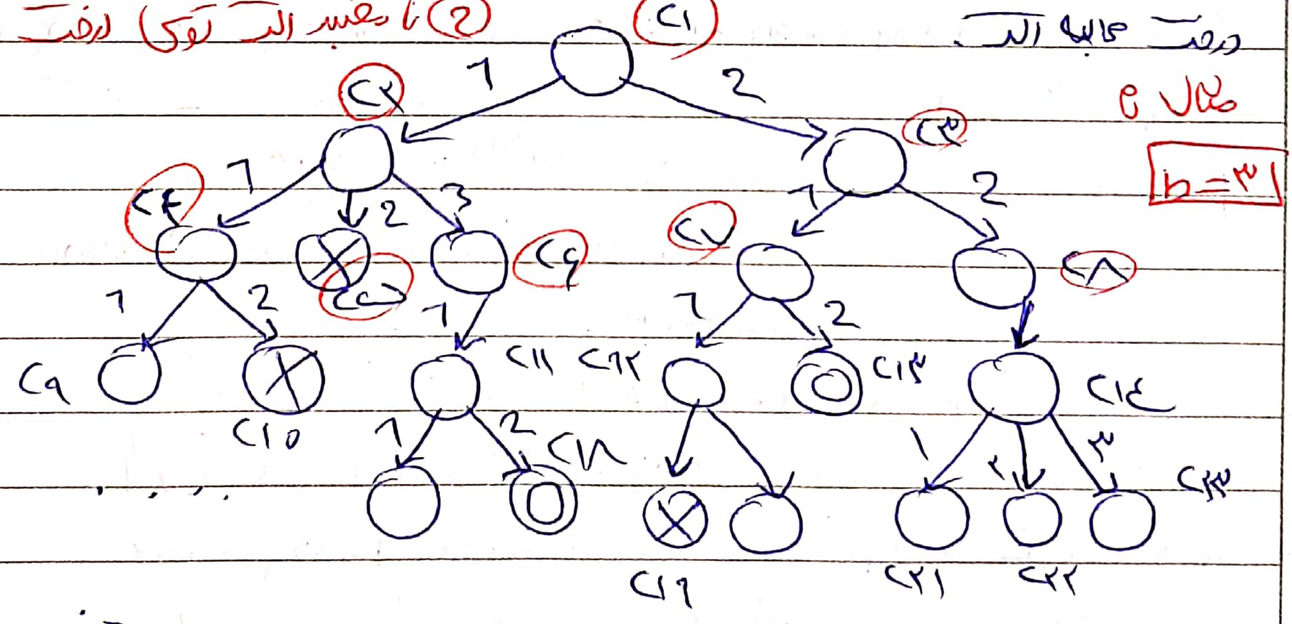
$b = \max|\delta(q,a)|$ max تعداد فرزندهای یک node

توی درخت محاسبه هر رشته ← ① یک node را مشخص می‌کنه یا

هر رشته که توی الفبای $\{1,2,\ldots,b\}^*$ آدرس یکی از node ها توی درخت محاسبه است. ② نامعتبر است توی درخت

مثلا e

$\boxed{b=3}$



آدرس‌ها e ← ٤ c1

c2 ← 1   c3 ← 2

c4 ← 11   c5 ← 12   c6 ← 13

c7 ← 21   c8 ← 22

هر node یک آدرس داره ~~۳۱~~ نداریم چون فرزند سوم نداریم

رشته‌های داخل مجموعه $\{1,2,...,b\}^*$ را به ترتیب lexicographic

ردیف می‌کنیم اگر ① آدرس معتبر باشد → کانفیگ accepting هست یا نه
② نامعتبر → رها می‌کنیم رشته را میریم سراغ بعدی

$b=3$ → lexicographical order of strings in $\{1,2,3\}^*$: ε, 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, ...

132 → چون تفریبش هیچ node ای نداریم → آدرس نامعتبر است → باید رها کنیم و بریم سراغ آدرس بعدی

قضیه: برای هر NTM ای که nondetermistic است & یک TM استانداردی داریم که معادلش است

اثبات: یک DTM مثل D می‌سازیم که معادل N است. ابتدا $b = \max|\delta(q,a)|$
تا وقتی که یک accepting config پیدا کند، D تولید می‌کند رشته‌های داخل $\{1,2,3,...\}^*$ را به ترتیب lexicographic یعنی تا سطح اول
هر آدرس فرزندِ یک node
درخت محاسبه

node تفریبش accepting هست یا نه؟ اگر رشته به فرزند به آدرس نامعتبر بود یا accepting نبود → آدرس بعدی را می‌سازد تا اینکه یک کانفیگ accept پیدا کند.

M یک TM که 3 tape دارد tape1 را دست‌نخورده نگه می‌داریم ورودی w را نگه میدارد
اولین مسیر چیه
w ورودی N است

tape2 → N را روی یک branch مشخص simulate می‌کند و tape3 هم برای تولید رشته‌های داخل مجموعه $\{1,2,...,b\}^*$ به ترتیب lexicographic
نوار ۲ را کپی کن توی branch ها
یک آدرس را مشخص می‌کند

مراحل:

① w را بنویس توی tape1 و tape2 را empty کن و ε را بنویس توی tape3

② کپی کن w را توی tape2

③ فرض کن α یک رشته‌ای است که روی tape3 قرار دارد

④ N را روی w simulate کن با توجه به آدرسی که α مشخص می‌کند

اگر مسیر را طی کردیم و مسیر به یک rejecting config ختم شد یا معتبر نیست ←

صبر کن رشته بعدی توی ترتیب lexicographic (→ به یک مسیر معتبر را مشخص نکرد یا به کانفیگ رسیدیم که accepting نبود) → برو به مرحله ۴

اما اگر به یک کانفیگ accepting رسیدیم ← accept می‌کنیم

مرحله ۴: e رشته‌ای که توی tape3 است را با رشته بعدی در ترتیب lexicographic جایگزین کن و برو به مرحله ۲

= n (گره‌ها) که جدا از n است

b = ۳ و ۱۱۲۳۱ توی tape3 بود → تبدیل به ۱۱۲۳۲ → ۱۱۲۳۳ → ۱۱۳۱۱ → ۱۱۳۱۲ → یعنی بعدی توی ترتیب lexico

اگر accepting بود ← halt می‌کنه، اما اگر رشته توی لوپ بیفته، طول می‌کشه

همه 2-PDA معادل یک TM است و قدرتمندتر از 1-PDA است

ولی اگر بیش از ۲ تا stack داشته باشیم، PDA ما قدرتش تغییر و افزایش نمی‌یابد

0-PDA ← همان NFA است که استک ندارد

1-PDA ← همون PDA معمولی

2-PDA ← قدرت TM را دارد

اگر توی PDA به جای داشتن یک stack یک queue داشته باشیم ←

همون PDA حامل برابر با TM است

Session 27 | Context sensitive languages — کلاس زبان‌های حساس به متن

تعلیم این کلاس از زبان‌ها ← هم ماشین داریم هم گرامر

LBA (linear bounded Automata)

LBA ← یک TM است که nondeterministic است و tape آن محدود است

است یعنی head نمی‌تواند از محدوده‌ای ورودی فراتر رود

محدود کردن حرکت head روی tape و tape ناصحدود است ولی حرکت محدود

[ ] کاراکتر های خاص برای مشخص کردن حدود حرکت head روی tape

head نمی تواند این دو کاراکتر را تغییر بده ← وقتی بهش برخورد می کنه

اگر به [بخوره نمی تونه بره چپ

اگر به] ← راست

$\boxed{q_0 [w]}$ کانفیگ اولیه

★ $\boxed{\text{NLBA}}$ ها

LBA ها ← الزاما nondetermistic هستند

عدم قطعیت دارن

کلاسی که توسط این ماشین ها تعریف میشه ← همون context sensitive language میشن

مقایسه ی قدرت NLBA ها و DLBA ها حل نشده است

مسئله

unrestricted grammers ← هیچ محدودیتی روی فرم rule ها برای شون وجود ندارد

تنها محدودیت ← سمت چپ هر rule ای باید یه variable داشته باشد

phrase structure

$G = (V, \Sigma, R, S)$

مجموعه variable ها — الفبا — مجموعه terminal ها — مجموعه rule ها — متغیر start — $\boxed{S \in V}$

فرم rule ها $(\alpha \to \beta)$

فرم خاصی ندارند

α و $\beta \in (V \cup \Sigma)^*$ و α حداقل یک متغیر است (حاوی)

مثال $aabAAB \to \varepsilon$ $\quad aAbb \to b$

کلاس زبان هایی که توسط این گرامرهای unrestricted توصیف می شود برابر است با کلاس زبان ← turing recognizable

recursively enumerable

turing decidable ها ← زبان هایی که تصمیم پذیرن turing decider داریم اما recognizable ها

کلاس: ← بزرگتر هستند که دربردارنده ی زبان های هستند که توسط TM ها recognize میشن

+ برای هر unrestricted grammer G ← یک TM الزاما T هست که $L(T) = L(G)$

برای هر TM با الفبای ورودی Σ ← یک unrestricted grammer هست الزاما G که زبان $L(T) \subseteq \Sigma^*$ هست را تولید میکنه

$\alpha \Rightarrow^*_G \beta$ β را می‌توان از روی α بدست آورد طی صفر یا بیشتر

گرامر

$L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^*_G w \}$

زبان تولید شده گرامر

مجموعه‌ای از رشته‌ها

گرامر unrestricted که S می‌تواند طی صفر گام یا بیشتر → اونها را تولید کند

ex

$L = \{ a^{2^k} \mid k \in Z \geq 0 \}$ همه رشته‌هایی که طولشان توانی از ۲ است و $\Sigma = \{a\}$ (الفبای یک حرفی)

$S \to LaR$ ۲ تا متغیر کنار هم $DR \to R$

$L \to LD$ $L \to \varepsilon$

$Da \to aaD$ $R \to \varepsilon$

یک متغیر ولی terminal → unrestricted grammer

D یک doubling operator است یعنی وقتی D از سمت راست یک a عبور می‌کند → تعداد a ها ۲ برابر می‌شه

سمبل R و D را حذف می‌کند یعنی وقتی D به R برخورد کند G کارش تمام می‌شه

$DR \to R$

مثال رشته‌ی G $aaaa$

$S \Rightarrow LaR \Rightarrow LDaR \Rightarrow LaaDR \Rightarrow$

$LaaR \Rightarrow LDaaR \Rightarrow LaaDaR \Rightarrow$

$LaaaaDR \Rightarrow LaaaaR \Rightarrow aaaaR \Rightarrow aaaa$

$L \to \varepsilon$

$R \to \varepsilon$

اگر رشته‌ای با طول بیشتر بخواهیم ← از این حالت باید به جای $\boxed{L \to LD}$ را تکرار کنیم و D دوباره از روی a ها رد شه و تعداد a ها را ۲ برابر کنه

$LaaaaR \Rightarrow^* LDaaaaR \Rightarrow$

$LaaaaaaaaDR \Rightarrow^*$

a — a

aFA

ex

$L(G) = \{ a^n b^n c^n \mid n \geq 0 \}$ گرامر G را پیدا کنید به طوری که

G → شامل S, B, C و rule های زیر است

(1) $S \to aSBC \mid \varepsilon$ (4) $CB \to BC$, (5) $bB \to bb$

(2) $aB \to ab$,

(3) $bC \to bC$ (6) $cC \to cc$

ای PDA هیچ ← نیست CFL ، $a^n b^n c^n$ زبان — قبلاً ثابت کردیم

گرامری که بتواند این زبان را توصیف کند (با لم تزریق ثابت کردیم)

از rule $S \to aSBC \mid \varepsilon$ استفاده می‌کنیم تا رشته‌هایی بدهیم

$a^n(BC)^n$ بسازد به همان تعداد که a می‌سازیم باید BC بسازیم بعد از

$CB \to BC$ استفاده می‌کنیم و تمام کنم های B ها را قبل از C ها — رشته حاصل

$a^n B^n C^n \to$ C ها را با b کوچک و B ها معادل تمام کنم

را با c کوچک جایگزین کنم — در اول $B \to b$ سوال

از این C ها نباید استفاده شود — $C \to c$ استفاده کردیم

ex rule (ساخت رشته) aaabbbccc

$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaSBCBCBC$

$\Rightarrow aaaBCBCBC \Rightarrow aaaBBCCBC \Rightarrow aaaBBCBCC \Rightarrow$

$aaaBBBCCC \Rightarrow aaabBBCCC \overset{*}{\Rightarrow} aaabbbCCC \Rightarrow$

$aaabbbcCC \overset{*}{\Rightarrow} aaabbbccc$ $\boxed{bC \to bc}$

$bB \to bb$ قانون دیگر

ممکن است شبیه رشته‌ای ایجاد کرد که توش از غیرپایانی نباشد ← جواب

مثال $aaaBCBCBC \Rightarrow aaabcbcbc$ ✗

درحالی که این رشته نباید توش زبان تعریف شده باشد — منظور از این rule ها ممنوع است

ex۳ $L(G) = \{ww \mid w \in \{a,b\}^*\}$ → non CFL — با لم تزریق اثبات کردیم

$V = \{S, T, A, B, R, L_a, L_b, [,]\}$

$S \to T]$. $T \to aTA \mid bTB \mid [R$ — T می‌تواند از دو طرف

$RA \to AR$ $RB \to BR$ به طور همزمان رشد کند

$AR] \to L_a]$ $BR] \to L_b]$

$AL_a \to L_aA$ $AL_b \to L_bA$

$BL_a \to L_aB$ $BL_b \to L_bB$ وقتی T در

$[L_a \to a[R$ $[L_b \to b[R$ — سمت چپ قرار بگیرد

$[R] \to \varepsilon$ — جای خودش را به — ساز را ثابت

T — جای خودش را به R می‌دهد — $[R$ می‌دهد — reverse کند R, A

$S \Rightarrow T] \overset{*}{\Rightarrow} abbTBBA] \Rightarrow abb[RBBA]$

اگر قبل از [R ← A باشه a ← [a

[b ← B ← 〜 ← 〜

[a ← جای را با هر A و هر B عوض می‌کنه و خودش را می‌رسانه به [

وقتی رسید به [← تبدیل به یک a خارج از [می‌کنه

[b هم همین‌طور

در نهایت [R باقی می‌ماند ← به جایش ε می‌گذاریم

مثال aabaaaba → اول w بعد w معکوس ($w^{reverse}$)

$S \Rightarrow T] \overset{*}{\Rightarrow} aabaT[ABAA]] \Rightarrow aaba[RABAA] \overset{*}{\Rightarrow}$
$aaba[ABAAR] \Rightarrow aaba[ABAL_a] \overset{*}{\Rightarrow} aaba[L_aABA] \Rightarrow$
$aabaa[RABA] \overset{*}{\Rightarrow} aabaa[ABAR] \Rightarrow aabaa[ABL_a]$
$\overset{*}{\Rightarrow} aabaa[L_aAB] \Rightarrow aabaaa[RAB] \overset{*}{\Rightarrow}$
$aabaaaba\boxed{[R]} \Rightarrow aabaaaba$

ε

[a برمی‌گرده عقب و خودش را به [می‌رسانه

$AL_a \rightarrow L_aA$ $\qquad$ $[L_a \rightarrow a[R$ قبل اول

$BL_a \rightarrow L_aB$

Context sensitive grammer ← CSG

اگر گرامر unrestricted باشه که هیچ کدام از قاعده‌هاش کوتاه‌کننده طول نیستند ← ① $\alpha \rightarrow \beta \in |\beta| \geq |\alpha|$ ②

$\alpha, \beta \in (V \cup \Sigma)^*$

نمی‌توانیم ε را داشته باشیم ← نمی‌تونیم ε بسازیم با این گرامرها

زبان حساس به متن ← اگر یک گرامر حساس به متن وجود داشته باشه که

$\rightarrow L(G) \cup \{\varepsilon\} = L$ یا $L(G) = L$

یا یک زبان حساس به متن است اگر یه گرامر CSG بتوانه زبان L منهای ε

را توصیف کنه ← به این زبان حساس به متن می‌گیم یا زبان L

CSG ها نمی‌توانند ε داشته باشند

CF

ارتباط بین LBA ها و CSG ها ←

① اگر یک زبان توسط یک LBA، accept شود، یک CSG وجود دارد که می‌تواند آن زبان را بسازد ← قدرت CSG معادل قدرت LBA هاست

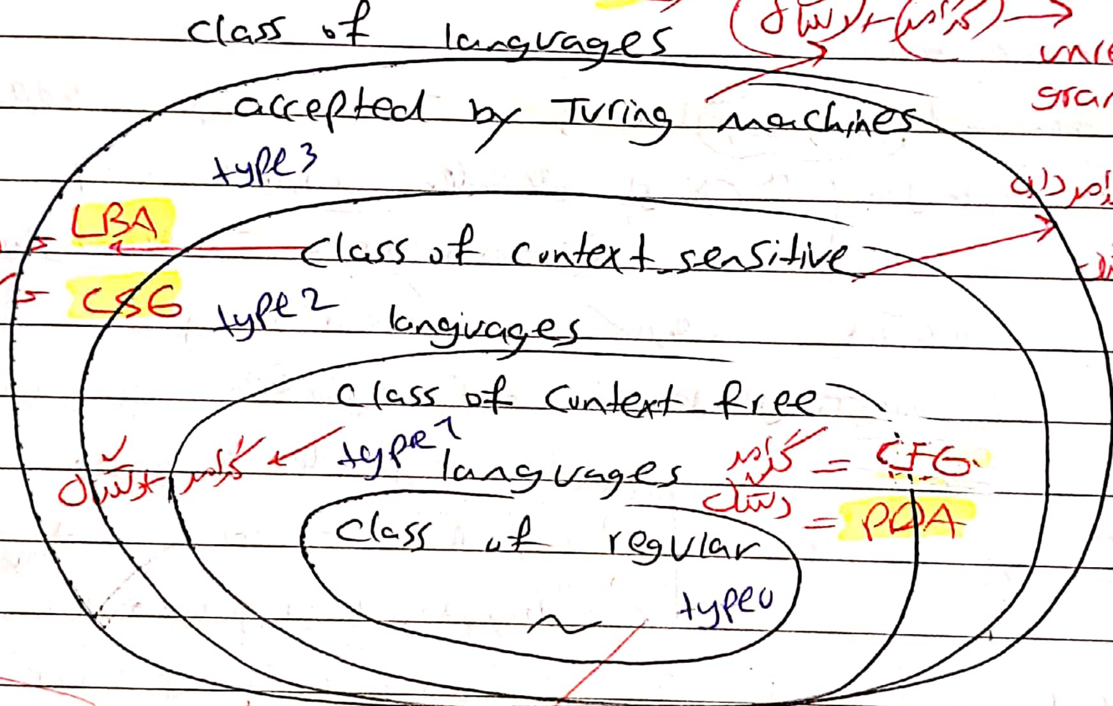
② اگر یک زبانی CSG باشد (زبان تولید شده توسط CSG توصیف شود) ← یک LBA هست که بتواند کل رشته‌ها را توصیف کند آن زبان را

LBA می‌تواند کل رشته‌ها را توصیف کند اگر وجود داشته باشد

یک زبان CFG الف ⟷ یک LBA تصمیم این زبان وجود داشته باشد

$\boxed{\text{CSL}}$ ← زیر زبان‌های turing decidable قرار دارند

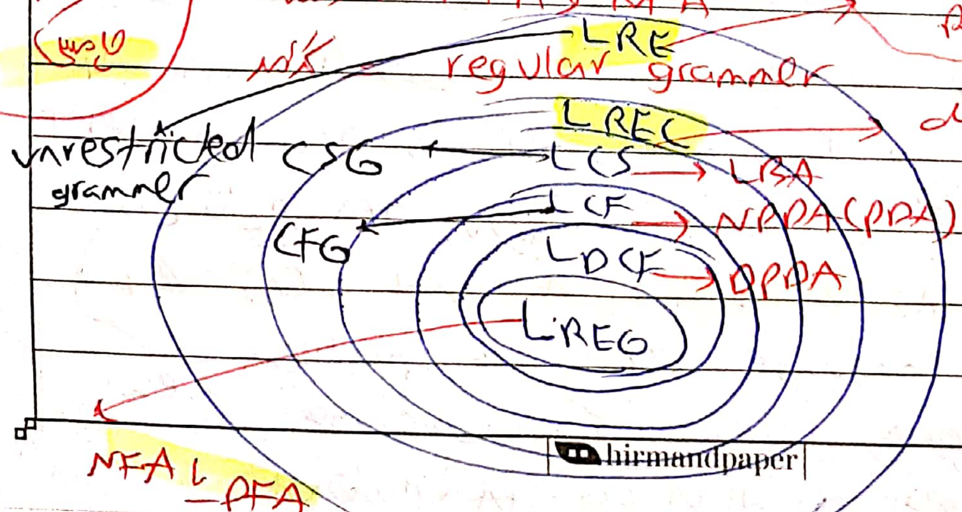
یک زبان recursive وجود دارد که CSL نیست



هم گرامر هم ماشین

ماشین = DFA و NFA

گرامر regular grammer



Turing recognizable ها
Recursively enumerable

deciders یا turing decidable

NFA یا DFA

* ورودی یک TM باید یک رشته باشد ← اگر یک object از طرف داریم باید encode کنیم جوری که رشته و اون رشته را به TM بدهیم

(صورت بندی)

session 23

گام اول برای تعیین حل پذیری یا حل ناپذیری یک مساله ← مشخص کنیم صورت مساله

problem view vs language view

بررسی به شکل زبان $\equiv$ (معادل) بررسی به شکل مساله مطرح شود

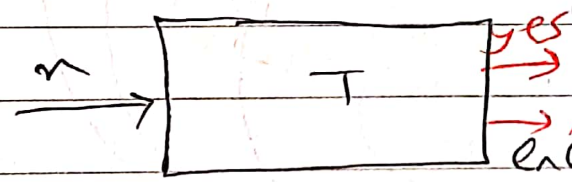
problem view: given a TM M & a string w
is $w \in L(M)$? که w توی زبان تعلق M هست؟

language view:

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

* وجود یک decider برای زبان T مثل T معادل وجود یک الگوریتم برای مساله الف و بررسی (دو مفهوم)

(توصیف کننده‌ی یک TM در کنار هم ← یا رشته الف رشته مثل w) و w داخل زبان TM الف



وجود یک decider مثل T برای A_{TM} ← یعنی T یک رشته مثل n را از ورودی می‌گیرد با توجه به encoding که داره می‌رسید آیا n داره یک TM + w را توصیف می‌کند یا نه؟

اگر n یک رشته معتبر نباشد ← NO برمی‌گرداند اگر رشته‌ی یک توصیف معتبر باشد ولی ساختن TM بنا کنیم ← NO برمی‌گرداند

تنها در صورتی که ① ورودی‌ها معتبر باشد ② w داخل $L(M)$ باشد ← yes برمی‌گرداند

ex1: یک DFA را به عنوان ورودی بگیر و تست کن که زبان تعلق این DFA تهی هست یا نه؟ ← emptiness

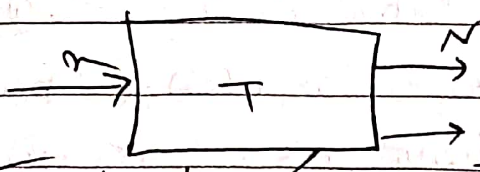
$E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

* problem view ← language view و زبان در بردارنده‌ی همه رشته‌های هست که این رشته‌ها یک DFA را توصیف می‌کنند

زبان تعلق این DFA‌ها $\emptyset$ الف ← وجود یک decider مثل T

وجود decider برای زبان ⟸ وجود الگوریتم برای مسئله

برای زبان ← به معنای وجود یک الگوریتم برای حل این مسئله است



اگر رشته ورودی معتبر نباشد یا اگر معتبر باشد و

زبانی که برای DFA وجود دارد ∅ نباشد

معتبر باشد (DFA را) را توصیف کند و yes

و زبان تهی DFA ← تهی باشد

T یک decider برای زبان E_{DFA} است

این مسئله حل پذیر است ← الگوریتم داریم برای

کافیه یک الگوریتم ارائه کنیم (روند قدم به قدم) ← که ترجیح تورینگ تعیین می‌کند

به واسطه اینکه روند الگوریتمی وجود دارد ← زبان decidable است

آیا مسیری از start state به یک state نهایی وجود دارد یا نه

اگر حداقل یک مسیر وجود داشته باشد ← زبان تهی DFA ناتهی است

اگر هیچ مسیری نداشته باشیم ← تهی

الگوریتم: ورودی ⟨A⟩ و A یک DFA است

① start state را mark کن

② تا جایی که مرحله ۳

③ state هایی که از state های mark شده به اونها یال ورودی وجود دارد

اونها را هم mark کن (یعنی از start برای state ها مسیر داریم)

④ آیا حداقل یک final state وجود داشته باشد که mark شده باشد ← اگر حداقلش بود ← زبان ناتهی است ولی اگر همه

final state ها mark نشده باقی بمانند ← یعنی زبان تهی DFA = ∅ است

ex2 مسئله تهی بودن یک CFG

$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

← این مسئله یعنی ← یک گرامر (CFG) را به عنوان ورودی بگیریم ببینیم آیا زبان تولیدی

این گرامر تهی است یا نه → emptiness

به عبارت دیگر زبان E_{CFG} ← یک زبان است و این زبان شامل همه‌ی CFG هایی

است که زبان تولیدی تهی است ← همه CFG هایی که اینها را دارند را به صورت

رشته encode می‌کنیم و در نظر می‌گیریم این زبان اگر این زبان یک decider

مثلاً T داشته باشد ← این decider یک رشته را به عنوان ورودی می‌گیرد اگر اون رشته

معین نباشد یا معین باشد و یک CFG داشته باشیم که زبان تهی‌اش نا تهی است ← No برمی‌گرداند

اگر معین باشد و نشان دهنده ... باشد که ... تهی است

روش دوم به کمک یک الگوریتم مثل CYK ساده تست کنیم آیا این رشته ... yes

داخل زبان L(G) قرار دارد یا نه و (تمام رشته‌های ممکن را ساده تست کنیم آیا داخل زبان

تهی هست یا نه چون ... آیا رشته‌ای را می‌پذیرد

اگر زبان یک decider مثل T داشته باشد

$n \rightarrow T \rightarrow$ No / yes

می‌کند و تولید می‌کند loop

یک الگوریتم برای مساله وجود دارد

این روش همان است

No برمی‌گرداند ولی هیچ وقت yes

روش دوم ← همواره halt نمی‌کند واقع حلقه ... برمی‌گرداند ← خوب نیست

یا yes یا No برگرداند

باید چک بشه آیا سمبل start ، generating است یا نه ← چون اگر

variable های generating مشخص کنند باید بشه آیا start generating است

یا نه ؟

1) همه terminal symbol ها را mark کن

2) تا وقتی که هیچ variable جدیدی mark نشده ؟

3) هر متغیر A ای که یک rule به شکل $A \rightarrow V_1V_2 \ldots V_k$

داره و هر سمبل V_1 ، V_2 ، ... ، V_k آن قبلا mark شده را mark کن

4) اگر start variable (هنوز mark نشده) ← accept کن

در غیر این صورت reject کن

یعنی از S هیچ string ای نمی‌توان تولید کرد

Σ^* باشد

اگر S generating باشد ← یعنی حداقل یک رشته توی Σ^* را می‌تواند تولید

کند ← زبان تهی G نا تهی است

← چون روند الگوریتمی را برای تست تهی بودن زبان تهی گرامر G داریم

مساله تست تهی بودن زبان تهی CFG ← حل پذیر است ← زبان E_{CFG}

decidable است و تست تهی بودن یا نبودن زبان تهی CFG

در صفحه قبل روش الگوریتمی قرار دارد.

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ ex۳

E_{TM} ← undecidable است ← الگوریتمی برای حل این مسئله وجود ندارد

چطوری ثابت کنیم این مسئله حل ناپذیر است؟

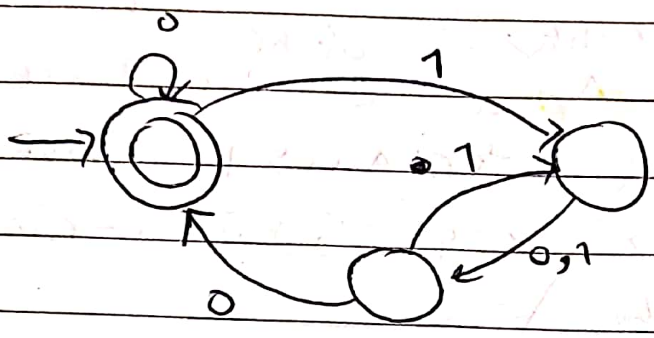
E_{DFA} ✓ حل‌پذیر است

E_{CFG} ✓ حل‌پذیر است

E_{TM} ✗

E_{LBA}

تمرین



∈ M

a. Is $\langle M, 0100 \rangle \in A_{DFA}$?

b. Is $\langle M, 011 \rangle \in A_{DFA}$?

c. Is $\langle M \rangle \in A_{DFA}$?

d. Is $\langle M, 0100 \rangle \in A_{REX}$?

e. Is $\langle M \rangle \in E_{DFA}$? → NO ← چون زبان M تهی نیست

f. Is $\langle M, M \rangle \in EQ_{DFA}$? → YES

یعنی معادل بودن دو DFA است ← آیا $L(M_1) = L(M_2)$ هست؟

رشته‌ی $\langle M, M \rangle$ توی زبان EQ_{DFA} هست

Undecidability

مسائلی وجود دارند که با کمک الگوریتم نمی‌توانیم اونا را حل کنیم (کامپیوترها از پس حلش برنمیان) مثل مسائلی
software verification → **undecidable**

اثبات حل‌ناپذیری A_{TM}

Acceptance

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

باید تست کنیم آیا رشته‌ی w داخل زبان تعریف ماشین تورینگ M قرار داره یا نه؟

برای Acceptance ← A_{DFA} و A_{CFG} هم داریم
decidable ✓ decidable ✓

باید چک کنیم $\{\langle DFA, w\rangle\}$
آیا DFA رشته‌ی w را قبول می‌کند یا نه؟

$\langle CFG, w\rangle$
باید چک کنیم آیا CFG، w را می‌سازد یا نه؟

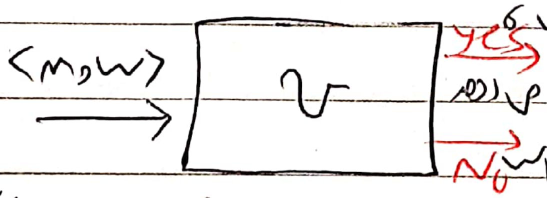
$\boxed{\text{الگوریتم CYK}}$ → decidable

* اگرچه A_{TM} Turing decidable نیست ولی Turing recognizable هست (TM زمانی recognizer یک مسئله است)

(A_{TM} الف)

U = یک TM الف که روی ورودی $\langle M, w\rangle$ که M یک TM الف و w یک رشته

① M را روی ورودی w، simulate کن
اگر M وارد accept state بشه ← accept کن
reject ← reject



U تنها کاری که می‌کنه M را روی w اجرا می‌کنه (سه الگوریتم ورودی‌اش به هم داده) تا ببینه M چکار می‌کنه، اگر M روی w متوقف شد ← yes برمی‌گردانه و اگر M روی w متوقف شد و reject کرد no برمی‌گردانه
① U ② U، no برمی‌گردانه
ممکن است M روی w، دچار حلقه شه ← U نمی‌تواند چیزی برگرداند ③
پس U یک decider نیست ولی recognizer هست.

مجموعه‌های همه‌ی TMها ممکن ← یک مجموعه‌ی شمارا است ← یعنی می‌شه اعضای این مجموعه

Diagonalization method را لیست کرد

برای مقایسه کردن ۲تا مجموعه که هر دو نامتناهی هستند باید چیکار کردیم

مثال: ۱ مجموعه ← مجموعه همه اعداد زوج

۲ ← همه رشته‌های کسری اعشاری ۰ و ۱ مقدار دارند

حالا اندازه‌ی ۱ و ۲ را باهم مقایسه کنیم هردو نامتناهی هستند

می‌گیم باید تابع یک به یک و پوشا بین ۲ مجموعه برقرار کنیم ← اگر بشه این کار را کرد یعنی ۲ مجموعه هم اندازه هستند اما اگر نشه ← هم اندازه نیستند

one to one: $a \neq b \rightarrow f(a) \neq f(b)$

$b \in B \rightarrow a \in A \quad f(a) = b$

مجموعه‌های A و B را می‌گیم هم اندازه هستند اگر یک تابع یک به یک و پوشا مثل f بین A و B وجود داشته باشد

$f: A \rightarrow B$

correspondence می‌گن

$E \quad N$

مثال: مجموعه‌ی اعداد طبیعی و اعداد زوج طبیعی ← هم اندازه هستند چون

mapping N to E $\quad \boxed{f(n) = 2n}$ یک تابع یک به یک و پوشا داریم

enumerable

(مجموعه شمارش‌پذیر یا شمارا countable)

اگر مجموعه متناهی باشد یا سایز اندازه‌اش با مجموعه N (مجموعه اعداد طبیعی) باشد

می‌شه اعضای این مجموعه‌ها را لیست کرد

مثل E (مجموعه همه اعداد زوج) ← چون با N هم اندازه است ← شمارا است

می‌شه اعضاش را لیست کرد

مجموعه‌ی Q^+ (اعداد گویا مثبت) $\quad Q^+ = \{\frac{m}{n} \mid m, n \in N\}$

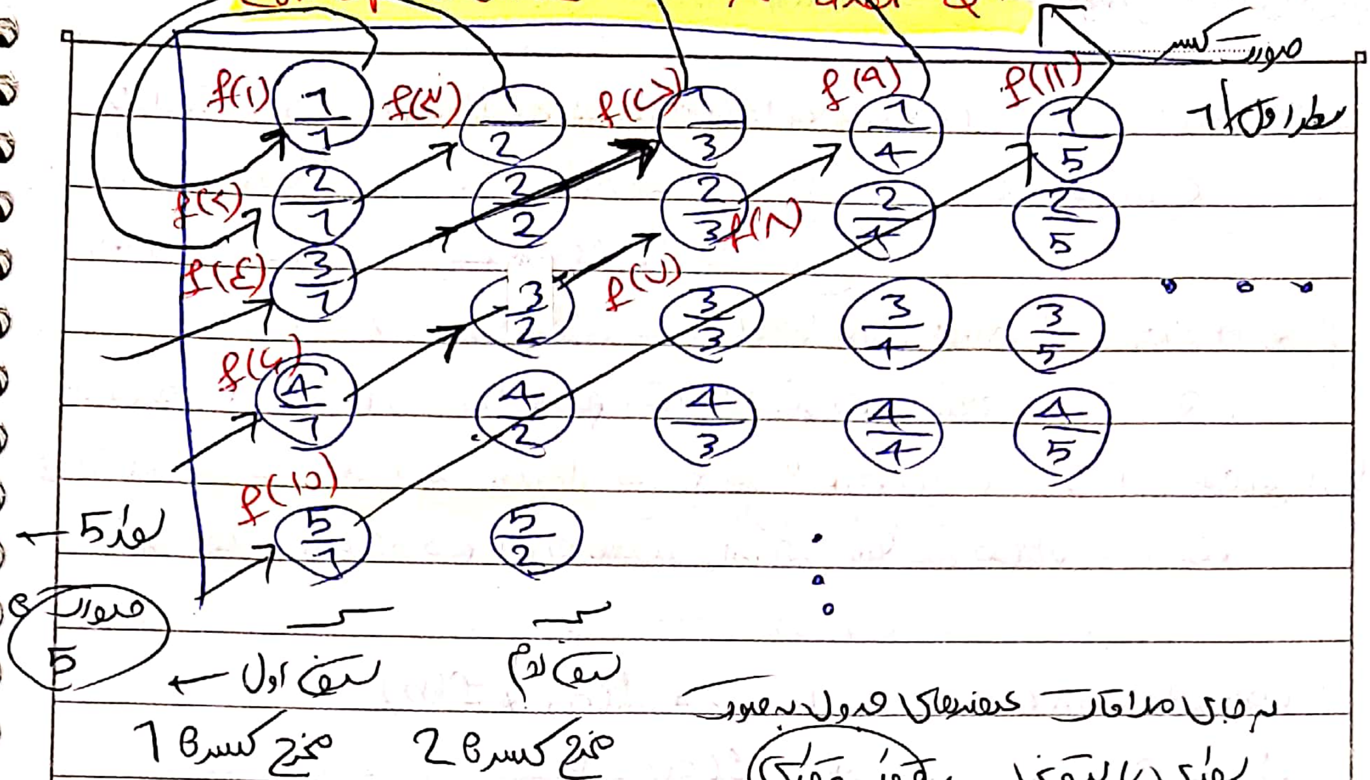
شمارا است و می‌شه اعضاش را لیست کرد

بین Q^+ و N یک correspondence وجود دارد

اندازه مجموعه‌ی همه اعداد حقیقی $\neq$ اندازه N

R ← اعضای R را نمی‌شه لیست کرد $\quad R > N \quad f: N \nrightarrow R$

Correspondence of N and Q^+



به جای ملاقات کردن‌های جدول به صورت سطری یا ستونی ← قطر قطری

میایم می‌بینیم اگر به درایه‌ای رسیدیم که این درایه قبلاً توی correspondence نقش داشته ← این را دیگه در نظر نمی‌گیریم

چون $\frac{1}{1} = \frac{2}{2}$ و $\frac{1}{1}$ قبلاً توی correspondence بوده ← از آن عبور می‌کنیم

به همین ترتیب اعضای N را با Q^+ مقایسه می‌کنیم

A مجموعه همه TM های ممکن ← مجموعه‌ای شمارا است

$\Sigma = \{0, 1, 2\}$ ، Σ^* مجموعه‌ای شمارا است

مجموعه همه رشته‌هایی که روی یک الفبای متناهی تعریف می‌شوند ← یک مجموعه شمارا است

$\Sigma^* = \{\epsilon, 0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, 000, \dots\}$

f(1) f(2) f(3) f(4)

پس یک تابع یک به یک و پوشا بین N و Σ^* وجود دارد ← اگر الفبای Σ متناهی باشد ← مجموعه همه رشته‌هایی که روی Σ وجود دارند شمارا است

با توجه به اینکه هر TM به نام M را می‌توان در بستر یک رشته توصیف کرد ← بعضی از رشته‌هایی که توی مجموعه Σ^* قرار دارند توصیف‌هایی برای TM ها خواهند بود و چون Σ^* شمارا پذیر است ← مجموعه همه TM های ممکن هم شمارا پذیر است چون TM را با encode کردن می‌توان در بستر یک رشته مثل <M> نمایش داد ← اگر فرایند encoding نیاز TM به الفبا را در بستر یک رشته روی الفبای Σ توصیف کند ← بعضی از رشته‌هایی که توی Σ^*

هستند ← یک TM را توصیف می کنند و بعضی دیگر هم توصیف نمی کنند ← چون

*Σ^* کاراکتر است و بعضی از اعضایش یک TM را توصیف می کنند ← مجموعه های TM ها هستند

ممکن هم نباشد رشته.

اثبات A_{TM} undecidable بودن A_{TM} مثل B برهان خلف

فرض خلف ← فرض می کنیم A_{TM} یک مسئله decidable است یعنی یک decider

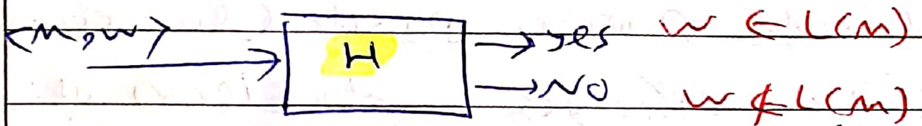
مثل H برای A_{TM} وجود دارد

H به عنوان ورودی توصیف یک TM در بستر یک رشته مثل w را می گیرد اگر

$w \in L(M)$ باشد (M و w را accept کند) ← H و yes برمی گرداند

اگر $w \notin L(M)$ ← (M و w را نپذیرد) ← NO

$$H(\langle M, w\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not } \sim \sim \end{cases}$$



به دو حالت است ← M و w را reject می کند

M و با خواندن w توی loop می افتد

از H استفاده می کنیم برای ساخت یک decider دیگر به نام D

ورودی D و یک رشته است که $\langle M\rangle$ را توصیف می کند

D و H را دارد ... یک TM

یعنی D و $\langle M, \langle M\rangle\rangle$ را ران می کند $\langle M, \langle M\rangle\rangle$

M توصیف M

به عنوان ورودی به H می دهد ← D می خواهد ببیند که آیا رشته $\langle M\rangle$ داخل زبان

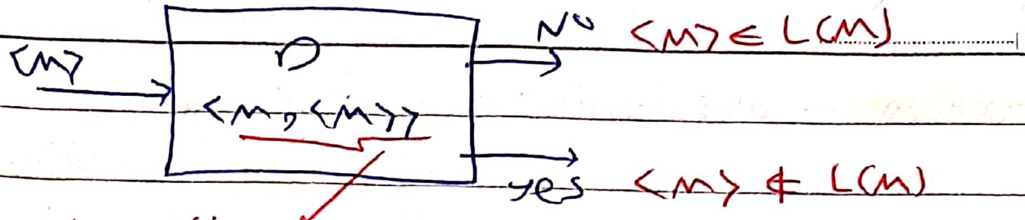
تشخیص M قرار دارد یا نه

اگر $\langle M\rangle$ توی M (زبان تشخیص M) قرار داشته باشد ← no برمی گرداند

$\langle M\rangle \in L(M)$

اگر $\langle M\rangle \notin L(M)$ ← yes برمی گرداند

چون H یک decider است ← D هم یک decider است

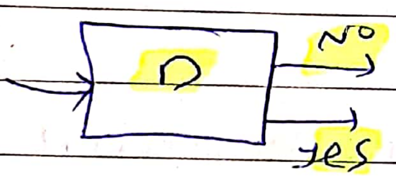


D به عنوان ورودی به H دادیم

$$D(\langle M\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \text{reject} & \text{if } M \text{ accept } \langle M\rangle \end{cases}$$

تناقض در جواب ورودی داریم ← اگر رشته‌ی توصیف کننده‌ی D را به عنوان ورودی به خودش بدهیم ← یعنی $D(\langle D\rangle)$ اگر $\langle D\rangle$ داخل زبان تصمیم D قرار داشته باشد ← D باید no برگرداند یعنی رشته‌ی $\langle D\rangle$ داخل زبان تصمیم D قرار نمی‌گیرد

اگر رشته‌ی $\langle D\rangle$ داخل زبان تصمیم D قرار نداشته باشد $\langle D\rangle \notin L(D)$ باید decider D و yes برگرداند ← رشته‌ی $\langle D\rangle$ داخل $L(D)$ قرار می‌گیرد ← تناقض ✗



if $\langle D\rangle \in L(D)$

if $\langle D\rangle \notin L(D)$

چون برای مسئله‌ی A_{TM} یک decider فرض کردیم به اسم H ←
فرض وجود H باعث ایجاد تناقض شد

D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$

← عجیب بود

session 24

★ برای اثبات undecidable بودن A_{TM} از زبان‌ها استفاده کنیم ←
اعضای مجموعه‌ی [illegible] های TM های ممکن را لیست کنیم ←
[illegible]

رشته‌های توصیف کننده TM

	⟨M1⟩	⟨M2⟩	⟨M3⟩	⟨M4⟩
M1	accept		accept	
M2	accept	accept	accept	accept
M3				
M4	accept	accept		

TM ها

مجموعه همه TM های ممکن را در قالب سطر و رشته‌های توصیف کننده TM ها را هم در ستون‌ها لیست می‌کنیم ←

یه جدول مثل روبرو داریم که هر سطر یک TM است و هر ستون یک رشته توصیف کننده یک TM

M1 یک TM، M1 رشته‌های توصیف کننده M1 را می‌پذیرد — M1 رشته توصیف کننده M2 را نمی‌پذیرد (reject, loop)

جدول ← می‌گه هر کدوم از TM ها کدوم یکی از رشته‌ها را accept می‌کنند کدوم را نمی‌کنند (یعنی loop یا reject می‌خورند)

فرض می‌کنیم یک decider مثل H برای A_{TM} وجود دارد ← خروجی H به ازای اینکه به عنوان ورودی (یه رشته و یک TM) را به H دهیم شکل زیر می‌شه

خروجی H ⇐ decider که الان

	⟨M1⟩	⟨M2⟩	⟨M3⟩	⟨M4⟩	
M1	accept	reject	accept	reject	
M2	~	accept	accept	accept	...
M3	reject	reject	reject	reject	
M4	accept	accept	reject	reject	
⋮		⋮			

مثلاً اگر ⟨M3, ⟨M3⟩⟩ را به عنوان ورودی به H بدهیم با توجه به اینکه توی سطر سوم M3 رشته‌ی ⟨M3⟩ را نمی‌پذیرد ← H، reject برمی‌گرداند

یا توی سطر M4 رشته‌ی ⟨M3⟩ را قبول نمی‌کند ← به دلیل reject یا loop

اگر ⟨M4, ⟨M3⟩⟩ را بدیم به H ← H، reject برمی‌گرداند

چون همه TM های ممکن را لیست کردیم ← D هم باید توی جدول ظاهر بشود. D رشته‌ی توصیف کننده یک TM را به عنوان ورودی می‌گیرد و اگر یک TM اون رشته را accept می‌کرد، D reject برمی‌گرداند و اگر رشته را reject می‌کرد

D ، accept برمی‌گرداند

	<M1>	<M2>	<M3>	<M4>	...	<D>
M1	accept					
M2		accept				
M3			reject			
M4				reject		
D	reject	reject	accept	accept		?

چون M3 ، توقف خودش یعنی <M3> را reject می‌کند → D ، accept می‌کند

چون M1 ، توقف خودش یعنی <M1> را قبول می‌کند → D ، reject برمی‌گرداند

* پس بعد قطری D ، نقیض (چیزی هست که روی قطر داره فاصله می‌گیره)
برای بدست آوردن بعد قطری D → کافی هست بشینم روی قطری خاصی نگاه و اون را نقیض کنم → درایه قطری بعد D بدست میاد

* وقتی بعد و سطر D الست باید درایه بالای ? اگر یک جاهایی accept فاصله شد → باید reject فاصله شه و اگر جایی reject فاصله شد یعنی باید accept فاصله شه →

TM _ D ، ورودی <D> را accept می‌کند اگر و فقط اگر D ورودی‌ای نقیض خودش را reject کند → تناقض منطقی ! → فرض خلف باطل <D> و فرض وجود یک decider برای A_{TM} ، رد می‌شه

الان فهمیدیم که A_{TM} ، undecidable است → اگر بخواهیم مسائل دیگر را اثبات کنیم می‌تونه از A_{TM} استفاده کنیم

Reducibility → کاهش‌پذیری

یک مسئله را به یک مسئله دیگر تبدیل می‌کنیم اگر مسئله‌ی اول ، undecidable باشد هم undecidable چون مسئله‌ی اول ، را ثابت می‌کند به مسئله دوم → یک جواب برای مسئله‌ی دوم می‌تونه برای حل مسئله‌ی اول استفاده کند

همواره ۲ تا مسئله داریم

اگر A کاهش‌پذیر باشد به B → می‌شه از یک جواب برای B

استفاده کنیم برای اینکه جوابی برای مسئله‌ی A بدست آوریم.

یعنی می‌شه A را حل کرد با داشتن جواب مسئله‌ی B

اگر B حل‌پذیر باشد (undecidable) B ⟵ A

چون روشی برای تبدیل A به B داریم و B decidable بوده، B یک روشی پیدا می‌کند به A

decidable بودن مسئله دوم ⟵ باعث می‌شود مسئله اول هم decidable شود (نتیجه می‌دهد)

و برعکس

اگر A undecidable باشد ⟵ B هم undecidable است (نتیجه می‌دهد)

مسئله‌ی $HALT_{TM}$

$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$

به عنوان ورودی یک TM و یک رشته می‌گیریم و باید تصمیم بگیریم آیا M روی w متوقف می‌شود یا نه و اگر halt می‌کند (با reject یا accept) باید accept برگردانیم و اگر توی loop می‌افتد ⟵ reject برمی‌گردانیم)

اثبات به کمک برهان خلف ⟵ فرض می‌کنیم مسئله‌ی $HALT_{TM}$ decidable است

فرض خلف

اگر قرار باشد که $HALT_{TM}$ decidable باشد ⟵ A_{TM} هم decidable خواهد بود

روند اثبات: برای $HALT_{TM}$ یک decider به اسم R داریم ⟵ با برعکس

از R برای A_{TM} یک decider طراحی کرد به اسم S

S یک ورودی به فرم ⟨M, w⟩ می‌گیرد

یک TM — یک رشته

⟨M, w⟩ را که ورودی S هست را به R می‌دهیم و اگر R decide کرد که M روی w متوقف نمی‌شود (دچار loop می‌شود) ⟵ S باید reject برگرداند ⟵ S باید به خوبی R نگاه کند

اما اگر R decide کرد که M روی w متوقف می‌شود یعنی halt می‌کند ⟵

چون w توی کوشش ماشین M شبیه‌سازی کرد

w را به M می‌دهیم تا اجرا کنه چون می‌دانیم دچار loop نمی‌شود ⟵

hirmandpaper

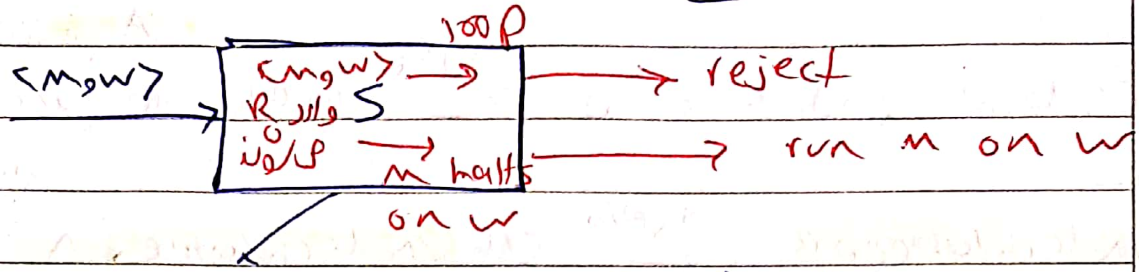
M را روی w ، run می‌کنیم و حاصل را برمی‌گردانیم (نتیجه‌ی M روی w را برمی‌گردانیم

اگر M ، w را بپذیرد ← accept برمی‌گردانیم

و اگر reject کرد ، reject برمی‌گردانیم ← وجود یک decider

برای $HALT_{TM}$ ، منجر به وجود یک decider برای A_{TM} می‌شود ←

تناقض با اینکه A_{TM} ، undecidable است



A_{TM} $\quad$ $R \rightarrow HALT_{TM}$

برای طراحی یک decider برای A_{TM} ← با بهره‌گیری از یک decider برای $HALT_{TM}$ ، که اسمش R است ←

S = روی ورودی <M,w> که M یک TM است و w یک رشته است :

① <M,w> را به عنوان ورودی به R می‌دهیم

② اگر R ، reject کرد (یعنی M روی w ، loop می‌شود) ←

ما باید reject برگردانیم (S ، reject برگرداند) چون w داخل L(M) نیست

به‌دلیل loop ، $w \notin L(M)$ ، $w \in L(M)$ نیست

③ اگر R ، accept کرد (یعنی R گفت که M روی w ، متوقف می‌شود)

ما M را روی w ، simulate می‌کنیم ← یعنی می‌دهیم w را به عنوان

ورودی به M بدهیم تا ببینیم M روی w چه کار می‌کند و مطمئن هستیم M روی w متوقف می‌شود

اگر M ، w را قبول کرد ← accept برمی‌گردانیم

~ reject ← ~ reject ~

کاری که می‌کنیم

☆ A_{TM} را reduce کردیم به مسئله‌ی $HALT_{TM}$

$A_{TM} \rightsquigarrow HALT_{TM}$

$E_{TM} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \emptyset\}$ exr

تست تهی بودن زبان تفسیر TM emptiness

به عنوان ورودی یک TM را می‌گیریم و باید decide کنیم آیا زبان تفسیر آن TM، تهی هست یا نه؟ undecidable است

E_{TM} ← دربردارنده‌ی همه‌ی رشته‌هایی هست که encodingهای TM هایی را توصیف می‌کنند که زبان تفسیرشان تهی است.

فرض خلف: فرض می‌کنیم که E_{TM} decidable است و یک decider مثل R برای E_{TM} وجود دارد.

نشان می‌دهیم با استفاده از R، می‌توان یک decider مثل S برای A_{TM} طراحی کرد، در حالی که با حقیقت اینکه A_{TM} undecidable است در تعارض است.

رشته‌ی ورودی S به صورت $\langle M, w \rangle$ ← encoding یک TM و یک رشته مثل w (یک TM، یک رشته)

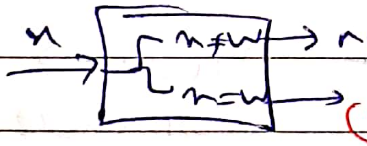
در داخل S می‌توان از R استفاده کرد

R می‌تواند تست کند که زبان تفسیر یک TM تهی هست یا نه؟

باید از قابلیت R استفاده کنیم تا S بتواند decide کند که w داخل L(M) هست یا نه؟

داخل S یک TM ای مثل M_1 طراحی می‌کنیم که M_1 هر رشته ورودی مثل x را که می‌گیرد چک می‌کند آیا x برابر w هست یا نه؟

اگر $w \neq x$ بود ← x را reject می‌کند M_1



اگر $x = w$ بود ← تازه simulate کردن شروع می‌شود

M روی w را برمی‌گرداند — اجرای M روی x (همان w) برگردان

اگر M، w را بپذیرد ← $L(M_1) = \emptyset$ یا $\{w\}$

زبان تفسیر M_1، w است — زبان تفسیر M_1

اگر M، w را نپذیرد ← M روی w loop کند یا w را reject کند

زبان تفسیر M_1 تهی است

$\langle M_1 \rangle$ را به عنوان ورودی به R می‌دهیم، اگر R accept برگرداند

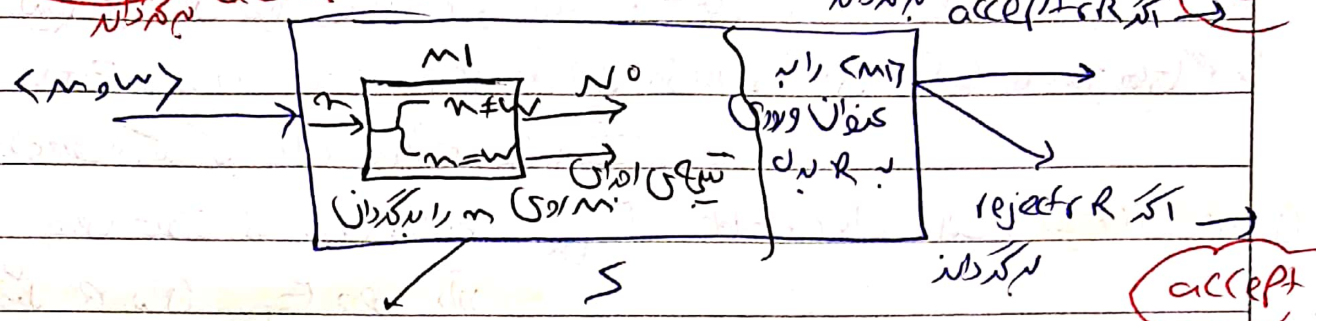
یعنی R نتیجه گرفته که $L(M_1) = \emptyset$ ← می‌فهمیم که M نتوانسته

w را accept کند ← ما هم S باید reject برگردانیم

اگر R reject برگرداند ← پس می‌فهمیم که $L(M_1)=\{w\}$ یعنی M نتوانسته w را

بپذیرد ← w داخل $L(M)$ است ($w \in L(M)$) ← باید accept برگردانیم

یعنی S باید accept برگرداند



$L(M_1)=\emptyset$ یا $\{w\}$

وقتی M نتواند w را بپذیرد ↓ $w \in L(M)$

(والبته reject ، loop)

وقتی M w را accept می‌کند ← M1 هم w را قبول می‌کند

یعنی: $w \in L(M) \Rightarrow \boxed{L(M_1)=\{w\}}$

اما وقتی که w توی زبان $L(M)$ نباشد ← M1 هم w را قبول نمی‌کند و

چون ورودی M1 درصورتی معتبر است که x = w باشد ← هیچ رشته‌ی دیگری را

قبول نمی‌کند (w هم که قبول نشد چون توی M نبود) ←

$\boxed{L(M_1)=\emptyset}$

$w \notin L(M) \Rightarrow L(M_1)=\emptyset$

پس وجود یک decider مثل R برای E_{TM} ← منجر به وجود یک decider

مثل S برای A_{TM} ← تناقض: چون اثبات کردیم که A_{TM} یک مسئله‌ی undecidable است.

A_{TM}

$HALT_{TM}$ E_{TM}

با کمک undecidable بودن A_{TM}

اثبات کردیم E_{TM} و $HALT_{TM}$ هم

حل ناشدنی هستند ← هرکدام از این دوتا برای مسائل دیگر استفاده کرد بجای

A_{TM} ← مثال: EQ_{TM} ← تست برابر بودن زبان‌های

دوتا TM تعیین equivalence آیا $L(M_1)\overset{?}{=}L(M_2)$ هستند یا نه

برای اثبات undecidable بودن

R.W

EQ_{TM} ← می‌شه از E_{TM} استفاده کرد و از حل ناپذیری E_{TM} نتیجه گرفت

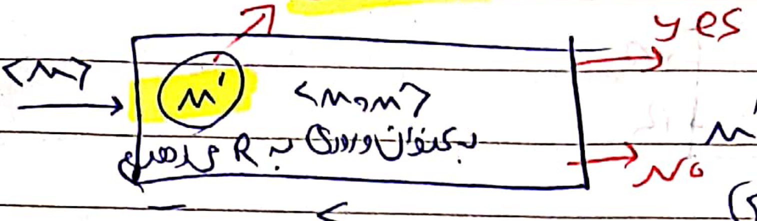
که EQ_{TM} هم undecidable است.

فرض خلف: فرض می‌کنیم EQ_{TM}، تصمیم‌پذیر باشد یعنی یک decider مثل

R برای EQ_{TM} وجود دارد ← از R می‌توان استفاده کرد یک decider برای

E_{TM} طراحی کنیم به اسم S

$L(M')=\emptyset$



داخل S یک TM دیگر به اسم M'

طراحی می‌کنیم که M' همه رشته‌های

ورودی را reject می‌کند ←

یعنی هستیم که $\boxed{L(M')=\emptyset}$

S برای E_{TM}

$\langle M,M'\rangle$ را به عنوان ورودی به decider ای می‌دهیم که برای EQ_{TM} داریم

اگر R تصمیم گرفت که $L(M)=L(M')$ است ← یعنی $\boxed{L(M)=\emptyset}$

پس باید S فرضی yes باشد (S هم yes برمی‌گرداند)

اگر R گفت که $L(M)\neq L(M')$ است ⟸ $L(M)\neq\emptyset$ ← کاملا

NO برمی‌گرداند

پس به کمک decider ای که برای EQ_{TM} داشتیم یک decider برای E_{TM}

هم داشتیم که این با undecidable بودن E_{TM} در تناقض است. (تناقض)

فرض خلف باطل → EQ_{TM} هم undecidable است

exe) اثبات کنید مسئله زیر تصمیم‌ناپذیر است

$SUBSET_{TM}=\{\langle M_1,M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs}$ و $L(M_1)\subseteq L(M_2)\}$

می‌توان از تصمیم‌ناپذیری EQ_{TM} استفاده کرد.

فرض خلف: $SUBSET_{TM}$ decidable است یک decider مثل

R وجود دارد مثل R ← می‌توان یک decider مثل S برای EQ_{TM}

طراحی کرد ← در بدنه S یک بار R را روی $\langle M_1,M_2\rangle$ اجرا کنیم

یعنی یک بار $\langle M_1,M_2\rangle$ را به R می‌دهیم و یک بار $\langle M_2,M_1\rangle$ را به

R می‌دهیم اگر R ورودی اول را accept کرد یعنی

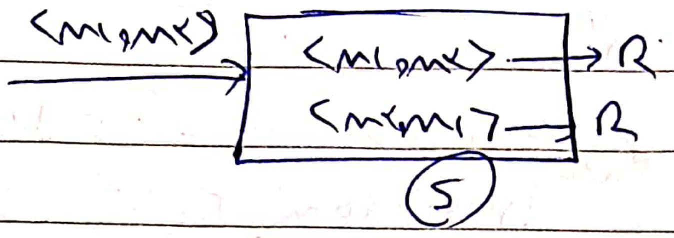
$L(M_1)\subseteq L(M_2)$

و اگر R ورودی دوم را هم accept کرد یعنی $L(M_2)\subseteq L(M_1)$ است

اگر هردوی ورودی ها accept برگردانند ← accept می دهیم از R

خروجی R و accept

ولی اگر حداقل یکی از اینها accept برنگردانند ← reject برمی‌گردانیم



راه دیگر ← از E_{TM} استفاده کنیم

به کمک R ($subset_{TM}$ decider) ← یک decider برای S می‌سازیم

برای E_{TM} می‌تونه decider کرد

توی S یک TM برای M_1 باید بسازیم که M_1 همه‌ی رشته‌ها را

ورودی را reject می‌کند ← $\langle M, M_1\rangle$ را به R می‌دهیم اگر

R accept برگرداند یعنی $L(M) \subseteq L(M_1)$ آنگاه ←

$\boxed{L(M)=\emptyset}$ ← S accept برمی‌گرداند

و اگر R reject برگرداند ← $L(M) \not\subseteq L(M_1)$ ←

$L(M) \neq \emptyset$ → S reject برمی‌گرداند

session 25 (مثال‌هایی از مسائل حل‌پذیر)

ex1) EQ_{DFA} یک زبان decidable است

تست معادل بودن دو DFA

ورودی B و A دو DFA ← $\langle A, B \rangle$ باید decide کنیم که

$L(A) = L(B)$ هست یا نه؟

(تفاضل متقارن)

$S_1 \Delta S_2 = (S_1 \cup S_2) - (S_1 \cap S_2) = (S_1 - S_2) \cup (S_2 - S_1)$

S_1 S_2

$S_1 = S_2 \iff S_1 \Delta S_2 = \emptyset$ پس ۲ مجموعه مساوی هستند

اگر و تنها اگر

اگر و تنها اگر تفاضل متقارن آن‌ها تهی باشد.

از روی A و B که دو DFA هستند یک DFA مانند C می‌سازیم که

$L(C) = L(A) \Delta L(B)$ ← باید تست کنیم آیا زبان C تهی هست یا نه؟

regular مطمئن هستیم این زبان است چون

$L(C) = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})$

و مکمل، اشتراک، و اجتماع بسته است ← زبان تعریف شده DFA

زبان‌ها

$L(A) \Delta L(B)$

با توجه به خواص بستاری و اینکه مطمئن هستیم کلاس زبان‌های regular تحت این ۳ عمل بسته است ← مطمئن هستیم که Δ دو زبان regular، regular می‌باشد ← تعریف $L(A) \Delta L(B)$ یک DFA داریم مثل C ←

باید چک کنیم $L(C)$ تهی هست یا نه؟

که E_{DFA} میشه که چون یک decider برای

تست تهی بودن DFA ها داریم

پس کافی است DFA C را به عنوان decidable ورودی به decider E_{DFA} بدیم ← اگر decider گفت $L(C) = \emptyset$

است ← $\boxed{accept}$ برمی‌گردانیم

اگر گفت $L(C) \neq \emptyset$ ← reject برمی‌گردانیم ←

در کل decider ای که برای EQ_{DFA} داریم از E_{DFA} استفاده کردیم

hirmandpaper

اثبات رکی G

decider برای EQ_{DFA}

F: یک decider B است که روی ورودی ⟨A,B⟩ اجرا می‌شود که A و B DFA است

1. C را به شکلی که گفته شد بساز DFA
2. T را یک decider برای E_{DFA} در نظر بگیر و T را روی ورودی ⟨C⟩ اجرا کن

یعنی $L(C)=\emptyset$

3. اگر T accept کند ما هم accept می‌کنیم
" T reject " " " reject "

$L(C)\neq\emptyset$

آیا EQ_{CFG} هم decidable است؟ یعنی آیا decider ای هست که به عنوان ورودی ۲ تا CFG را بگیرد و decide کند که آیا $L(G)=L(H)$ یا نه؟

$$EQ_{CFG} = \{\langle G,H\rangle \mid G \text{ and } H \text{ are CFG and } L(G)=L(H)\}$$

خیر نمی‌شود چون کلاس زبان‌های context free تحت عملگرهای اشتراک و مکمل بسته نیست ← EQ_{CFG} undecidable است

چون تضمینی نیست که $L(G)\Delta L(H)$ یک CFL باشد ← تضمینی وجود ندارد یک CFG مثل G برای توصیف این زبان وجود داشته باشد

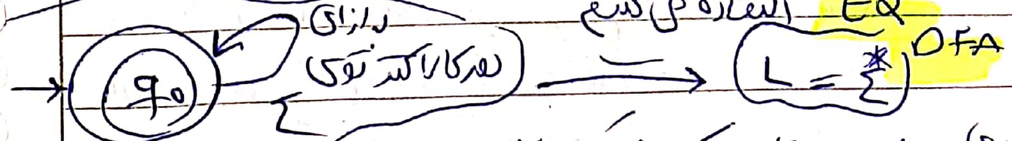
$$ALL_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A)=\Sigma^*\}$$ ex)

نشان دهید ALL_{DFA} decidable است.

یک DFA را به عنوان ورودی می‌گیریم و decide می‌کنیم آیا زبان تعریف این DFA برابر Σ^* هست یا نه؟

باید T decider بسازیم — ایده اول

1. یک decider به اسم M1 که ... M1 از یک decider دیگر به اسم T برای EQ_{DFA} استفاده می‌کنیم



M1 روی ورودی ⟨A⟩ جایی که A یک DFA است:

1. یک DFA مثل B می‌سازیم و مطمئن هستیم که $L(B)=\Sigma^*$ ←

start state B را برابر accept state B قرار می‌دهیم یعنی

② اگر T یک decider برای EQ_{DFA} باشد ← T را روی ⟨A, B⟩ اجرا کن

③ اگر T accept برگرداند یعنی $L(A)=L(B)$ ← $\boxed{L(A)=\Sigma^*}$ ←

M1 accept برمی‌گرداند

اگر T reject برگرداند ← $L(A)\neq L(B)$ ← $L(A)\neq\Sigma^*$ ←

M1 reject ←

روش دوم :

یک decider به اسم M2 ورودی ⟨A⟩ است که A یک DFA است

① یک DFA به اسم B را طوری بساز که $L(B)=\overline{L(A)}$ با جابجا کردن

state های accept و nonaccept توی DFA، A

② یک decider برای E_{DFA} به اسم T بساز ← T را روی ⟨B⟩ اجرا کن

③ اگر T accept برگرداند یعنی $L(B)=\emptyset$ است ← یعنی

$L(A)=\Sigma^*$ → M2 accept برمی‌گرداند

و اگر T reject ← $L(B)\neq\emptyset$ ← $L(A)\neq\Sigma^*$ ←

M2 reject برمی‌گرداند

$\boxed{\text{decidable}}$

ex) $INFINITE_{DFA}=\{\langle A\rangle \mid A$ is a DFA and $L(A)$ is an infinite language$\}$

یک DFA از ورودی

می‌گوییم باید decide کنیم آیا $L(A)$ نامتناهی است یا نه ؟

یک decider به اسم I می‌سازیم که ورودی آن ⟨A⟩ است که A یک DFA است

① فرض کن k تعداد state های A است

② یک DFA مثل D طراحی کن که همه‌ی رشته‌هایی که طولشان k یا بیشتر است را

این D باید accept کند

③ یک DFA مثل M ← که $L(M)=L(A)\cap L(D)$

④ تست کن آیا $L(M)=\emptyset$ هست به کمک decider برای E_{DFA} به اسم T روی M

متناهی است

⑤ اگر T accept کرد یعنی $L(M)=\emptyset$ ← یعنی $L(A)$ متناهی است → reject کن

$L(A)$ نامتناهی است

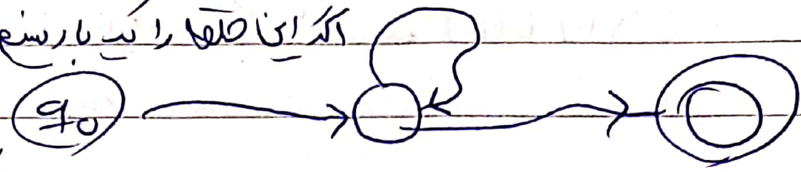
اگر T reject کرد ← یعنی $L(M)\neq\emptyset$ ← accept کن

رشته ای داریم که طولش k یا بیشتر است و تعداد state های A برابر k است

حداقل یک state ، دوبار دیده می شود (طبق اصل لانه کبوتری)

state ای است که بیش از یک بار دیده می شود

اگر این حلقه را یک بار برسیم به رشته ای می رسیم



اگر ۳ بار برسیم ← رشته طولانی تر

با ملاقات کردن → هر بار این حلقه را تکرار کنیم به رشته طولانی تری می رسیم →

این حلقه می تواند رشته هایی با طول دلخواه را ایجاد کنیم ← L(A) نامتناهی خواهد بود

تعداد state های A برابر k → یک DFA → A

یک DFA که زبانش تمامی رشته ها با طول ≥ k ← B

$L(A)$ نامتناهی $\longleftrightarrow L(A) \cap L(B) \neq \emptyset$ بیشتر است

ex

given a CFG G , a string n

is $L(G) = \{n\}$?

باید نشان دهیم این مسئله decidable است

است ← یک decider باید طراحی کنیم ← چون باید تعیین کنیم n

یا یک الگوریتم بنویسیم

گام ① تست کنیم آیا n داخل L(G) هست یا نه ؟ از A_{CFG} استفاده می کنیم الگوریتم CYK

② اگر $n \notin L(G)$ نباشد → $L(G) \neq \{n\}$ → reject کن

اگر $n \in L(G)$ باشد → باید تست کنیم آیا $L(G) = \{n\}$ است

③ $L(G) = \{n\} \iff L(G) \cap \{n\} = \emptyset$ است یا نه ؟

اشتراک L(G) با مکمل {n} تهی شود → $L(G) = \{n\}$

L(G) یک CFL است → تضمین یک PDA داریم

{n} هم یک زبان regular است → تضمین یک DFA داریم ←

می توانیم برای اشتراک یک PDA و یک DFA یک $\boxed{\text{PDA طراحی کنیم}}$

اشتراک یک زبان CFL و یک regular یک CFL است

برای اشتراک

در مرحله بعد اول PDA را به یک CFG تبدیل می‌کنیم ← باید چک کنیم که زبان تهی است یا نه

CFG G تهی هست یا نه و به کمک decider ای که برای E_{CFG} داریم

اگر زبان تهی G نبود ← یعنی $L(G)=\{n\}$ است ← accept

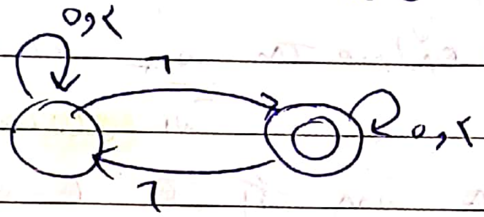
← G نبود ← $L(G) \neq \{n\}$ ← reject

مثل G ← $\langle G\rangle$ را به عنوان ورودی به decider ِ E_{CFG} می‌دهیم

EX $A = \{\langle M\rangle \mid M$ is a DFA that does not accept any string containing an odd number of 1s$\}$

آیا رشته‌ای که تعداد فرد یک داره در داخل $L(M)$ وجود داره یا نه

① یک DFA مثل O می‌سازیم که این DFA همه رشته‌هایی که تعداد فردی 1 هستند را قبول می‌کنه و بقیه را رد می‌کنه مثلاً $\Sigma=\{0,1,2\}$ الفبا



② یک DFA مثل B را ایجاد کن که $L(B) = L(M) \cap L(O)$

اشتراک ۲ تا زبان regular خودش regular است ← DFA O

③ چک می‌کنیم آیا $L(B)=\emptyset$ است یا نه و به کمک یک decider برای E_{DFA} ← $\langle B\rangle$ را به عنوان ورودی به یک decider برای E_{DFA} مثل T داریم می‌دهیم

④ اگر T accept برگرداند ← می‌فهمیم باید accept برگردانیم چون

در زبان تقاطع M و O رشته‌ای $L(M)\cap L(O)=\emptyset$ ←

که فرد تا 1 داشته باشه نداریم

اما اگر T reject برگرداند ← $L(M)\cap L(O)\neq\emptyset$ ← رشته‌ای توی

زبان M هست که فرد تا 1 داره ← reject

EX $\{\langle G\rangle \mid G$ is a CFG over $\{0,1\}$ and $1^* \cap L(G) \neq \emptyset\}$

ورودی یک CFG است روی الفبای باینری $\{0,1\}$ ← باید decide کنیم

آیا این گرامر می‌تواند یک رشته که تمام 1 را بسازد یا نه ← اگر توانست ← accept

اگر نه ← reject   زبان regular   زبان CFL

اشتراک یک CFL و یک regular ← CFL است

تبدیلش می‌کنیم به CFG داشته باشیم

① یک CFG مثل H طراحی می‌کنیم که $L(H) = \overline{T}^{*} \cap L(G)$

② تست می‌کنیم آیا $L(H) = \emptyset$ است یا نه؟ به کمک یک decider برای مسئله E_{CFG} مثل (R) ← $\langle H \rangle$ را به عنوان ورودی به R می‌دهیم

یک CFG ... CFG :

③ اگر R، accept برگرداند ← یعنی زبان تهی H، $\emptyset$ است ←
$L(G)$ نمی‌تواند یک رشته کامل T بسازد

اگر reject برگرداند ← یعنی $L(H) \neq \emptyset$ ← $L(G)$ می‌تواند
یک رشته کامل T بسازد ← $\boxed{\text{accept}}$ برمی‌گردانیم

$A_{DLBA} = \{\langle M, w \rangle \mid M \text{ is a DLBA that accept string } w\}$ $\boxed{A_{DLBA}}$

DLBA ← برعکس A_{TM} عدم decidable بودن TM است ← برخلاف A_{TM} که یک مسئله decidable است.

Lemma ۶: اگر M یک DLBA باشد و q تا state داشته باشد و
الفبای tape، g تا سمبل داشته باشد ← تعداد کانفیگ‌های متمایز که
M با روی tape، n خانه‌ای با آن کار می‌کند ← $\boxed{q n g^{n}}$ است که n طول رشته ورودی است

ادای رشته ورودی n طول است

DLBA ← شرط اضافه‌تری	$\lvert Q \rvert = q$ مجموعه state های Q
نسبت به TM، head را نمی‌تواند	M
از ورودی اش طرف‌تر ببرد	$\lvert \Gamma \rvert = g$ الفبای tape، Γ

یک کانفیگ، وضعیت ماشین را مشخص می‌کند

محتوای tape و head کجاست و در چه state ای

$q \times n \times g^{n}$

تعداد حالات state برای حالت فعلی — تعداد موقعیت‌هایی که head می‌تواند باشد — برای محتوای tape

n تا کاراکتر داریم هر کدام از این موقعیت‌ها می‌تواند g حالت داشته باشند.

tape

* تعداد کانفیگ‌های یک DLBA محدود است. (برای TM ها این طور نیست)

① یک decider است که در ورودی ⟨M,w⟩ را می‌گیرد که M یک DLBA است و w یک رشته است

② M را روی w اجرا می‌کند برای $|Q| n g^n$ step یا تا وقتی که halt کند ← اگر قبل از اینکه این تعداد کانفیگ را ملاقات کند halt کند با accept یا reject ← همین جواب را برمی‌گرداند

اگر تعداد step هایی که می‌رود از تعداد کانفیگ‌ها $> q n g^n$ شد ← یعنی داریم یک config را بیش از یک بار می‌بینیم ← توی loop افتادیم ← reject می‌کنیم

برای DLBA می‌شه وقوع loop را فهمید

مسئله‌ی E_{DLBA} undecidable است.
E_{TM}

session 26

post correspondence problem (PCP)

← نوع دومینو و از هر کدام بی‌نهایت داریم

سوال: آیا می‌شه این دومینوها را طوری کنار هم بگذاریم که رشته‌ای که بالا داریم با رشته‌ای که پایین داریم با هم برابر باشند؟

(a)

10	01	0	100	1
101	100	10	0	010

یک مثال از PCP که جوابش مثبت است

(b)

رشته بالا	10	1	01	0	100	100	0	100
رشته پایین	101	010	100	10	0	0	10	0

از این ۳ بار استفاده شده ← تکراری مجاز است

مثال ③ $\{[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]\}$

$[\frac{a}{ab}][\frac{b}{ca}][\frac{ca}{a}][\frac{a}{ab}][\frac{abc}{c}]$ رشته بالا = رشته پایین

ولی هیچ‌گاه اگر در هر دومینو طول رشته بالایی اکیداً بیشتر از طول رشته پایینی باشد ←

$\{[\frac{abc}{ab}], [\frac{ca}{a}], [\frac{acc}{ba}]\}$

تعریف رسمی PCP

یک نمونه از PCP یک set از (جفت) pair ها هست

$\{(\alpha_1,\beta_1),(\alpha_2,\beta_2),\dots,(\alpha_n,\beta_n)\}$ $n\geq 1$

α_i و β_i ها رشته‌های ناتهی هستند — α_i ها ← رشته‌های بالایی

آیا یک عدد صحیح مثبت k و دنباله‌ای مثل $i_1, i_2, \dots, i_k$ که $1 \le i_j \le n$ وجود دارد که — β_i ها ← پایینی

$\alpha_{i_1}\alpha_{i_2}\dots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\dots\beta_{i_k}$

آیا می‌توان طوری دومینوها را کنار هم قرار داد تا رشته‌های بالایی با پایینی برابر باشند

تکرار مجاز

PCP = {<P> | P is an instance of the post correspondence problem with a match}

← جوابش مثبت است

encoding instance هایی که جواب آن‌ها yes است

① تست تهی بودن اشتراک زبان‌های تولیدی دو تا CFG

دو تا CFG داریم ← آیا زبان تولیدی اشتراک این دو تا CFG تهی است یا نه؟ $L(G_1)\cap L(G_2)$ nonempty is

② تست مبهم بودن یک CFG

یعنی الگوریتمی برای تست مبهم بودن یا نبودن یک CFG نداریم

مسائل undecidable ای که در رابطه با CFG هستند به کمک PCB

تطبیق $\{(\alpha_1,\beta_1),(\alpha_2,\beta_2),\ldots,(\alpha_n,\beta_n)\}$

دو تا CFG ایجاد می‌کنیم ← یکی همه‌ی α_i ها و G_α

← β_i ها و G_β

گرامر تطبیق α_i ها و

$$\underbrace{S_\alpha}_{\text{start variable}} \to \underbrace{\alpha_i S_\alpha c_i \mid \alpha_i c_i \quad (1 \le i \le n)}_{2n \text{ تا rule دارد}}$$

α_i , c_i ← terminal ها

c_i ها ← سمبل‌های خارج از $\boxed{\text{الفبای } \Sigma}$ ← الفبای تطبیق α_i ها و β_i ها

G_β

گرامر تطبیق β_i ها و

$$S_\beta \to \underbrace{\beta_i S_\beta c_i \mid \beta_i c_i \quad 1 \le i \le n}_{2n \text{ تا rule}}$$

β_i و c_i ها terminal

اگر c یک رشته به طول k باشد که از سمبل‌های c_i تا c_{ik}

ایجاد شده باشد ← $c = c_{i1} c_{i2} \ldots c_{ik}$,

$(k \ge 1)$

دقیقا یک رشته مثل x داخل $L(G_\alpha)$ و

دقیقا یک رشته مثل y داخل $L(G_\beta)$ داریم که c، suffix ـِ x

است و c، suffix ـِ y هم است.

یعنی دقیقا یک رشته مثل x داخل زبان تطبیق G_α داریم که به c ختم میشه

~ G_β ~ y ~

$$x = \alpha_{ik} \ldots \alpha_{i2}\, \alpha_{i1}\, \underbrace{c_{i1}\, c_{i2} \ldots c_{ik}}_{c}$$

$$y = \beta_{ik} \ldots \beta_{i2}\, \beta_{i1}\, \underbrace{c_{i1}\, c_{i2} \ldots c_{ik}}_{c}$$

فرض کنید یه نمونه از PCB مثل I داریم ← به کمک I ، G و G_α و G_B را می‌سازیم ← یک نمونه مثل $\langle G_1, G_2 \rangle$ از مسئله‌ی

CFG Nonempty intersection

می‌سازیم و اسمش را $\boxed{F_1(I)}$ می‌گذاریم و

از روی I ، G رو که یک نمونه برای مسئله‌ی نسبت مبهم بودن باشه

یک گرامر می‌سازیم که زبان تعریف بکنه $L(G_\alpha) \cup L(G_B)$ ← این گرامر

در بردارنده‌ی rule های G_α و G_B به همراه ۲ تا rule (اضافه)

الف → $S \to S\alpha \mid SB$ → G بدین صورت

$L(G_\alpha)$ را می‌سازد کل $L(G_\alpha)$ — به کمک rule های G_B که توی الف — کل $L(G_B)$ را می‌سازد

G ← یک نمونه از مسئله‌ی نسبت ابهام الف و اسمش $\boxed{F_2(I)}$ الف.

اگر I یک yes از مسئله‌ی PCB باشه ← جواب $F_1(I)$ و $F_2(I)$ هم مثبت instance الف

یعنی اگر بشه i_1 ، B_{i_1} — $B_{i_{k-1}} B_{i_k} = \alpha_{i_1}$ — $\alpha_{i_{k}} \alpha_{i_{k-1}}$ ←

جواب $F_1(I)$ هم مثبت می‌شود چون رشته‌ای را هم داخل $L(G_\alpha)$

داریم هم $L(G_B)$

$\alpha_{i_k} \alpha_{i_{k-1}}$ — $\alpha_{i_1} C_{i_1}$ — $C_{i_k} =$

$B_{i_k} B_{i_{k-1}}$ — $B_{i_1} C_{i_1}$ — C_{i_k}

هم داخل $L(G_\alpha)$ هم $L(G_B)$

دو تا prefix هم برابر و دو تا suffix هم برابر

کل ۲ تا رشته برابر

پس ① yes بودن I ← yes بودن $F_1(I)$ را نشان می‌دهد

→ $L(G_\alpha) \cap L(G_B) \neq \emptyset$

② گرامر G مبهم الف چون رشته‌ای

هم داخل $L(G_\alpha)$ الف هم داخل $L(G_B)$ ← توی اشتراک

G مبهم می‌کنه

حتماً گرامر الف ← توی گرامر G ، G این رشته را که هم به کمک

$S\alpha$ و SB الف هم

$S \Rightarrow S\alpha \stackrel{*}{\Rightarrow} \alpha_{i_k} \alpha_{i_{k-1}}$ — $\alpha_{i_1} C_{i_1}$ — C_{i_k} این دو تا هم که

$S \Rightarrow SB \stackrel{*}{\Rightarrow} B_{i_k} B_{i_{k-1}}$ — $B_{i_1} C_{i_1}$. C_{i_k} — C_{i_k} برابر هستند

اگر $F_1(I)$ یک yes instance بشود برای تست ناتهی بودن اشتراک ← I هم

یک yes instance می‌شود.

اگر $F_2(I)$ هم ـــ ـــ مبهم بودن گرامر ـــ ـــ

برای یه رشته‌ای توکن $x \in \Sigma^*$ و یه دنباله‌ای $i_1 \dots i_k$ ، $i_1, i_2, \dots$

رشته‌ای مثل $x c_{i_1} c_{i_2} \dots c_{i_k}$ داریم که داخل اشتراک زبان تولید شده و

زبان تولید شده توسط GB است ← رشته‌ای که راهی که هم به کمک S_α ساخته هم

SB ← قاعدتاً (الزاماً) باید فرمش به شکل زیر باشد چون هم باید توسط S_α

ساخته بشود و هم توسط SB

$$\alpha_{i_k}\alpha_{i_{k-1}} \dots \alpha_{i_1} c_{i_1} \dots c_{i_k} = \beta_{i_k}\beta_{i_{k-1}} \dots \beta_{i_1} c_{i_1} \dots c_{i_k}$$

$x \in \Sigma^*$ توسط S_α ساخته شده ؛ $x \in \Sigma^*$ توسط SB ساخته شده

پس با مساوی قرار دادن suffix یک‌ها که مشترک برابر هستند

$$\alpha_{i_k}\alpha_{i_{k-1}} \dots \alpha_{i_1} = \beta_{i_k}\beta_{i_{k-1}} \dots \beta_{i_1}$$

پس جواب نمونه‌ای PCP هم yes خواهد بود

همان I

undecidable بودن مسائلی PCP ← تسری پیدا کرد به ۲ تا مسأله

یک نمونه از مسئله تست ناتهی بودن ← $F_1(I)$

اشتراک زبان‌های تولید ۲ تا CFG

I

یک نمونه از PCP

یک نمونه از مسائل تست ابهام ← $F_2(I)$

جواب $F_1(I)$ مثبت باشد $\Longleftrightarrow$ I جوابش مثبت باشد

" " $F_2(I)$ " " $\Longleftrightarrow$ " "

اگر و تنها اگر

برهان خلف ←

& decider بسازیم برای $F_2(I)$ ← با این I که نمونه‌ای از PCP است

حل کنیم ← تناقض

برای یک نمونه داده شده از مسئله PCP به نام I ← $F_2(I)$ را اجرا می‌کنیم

به decider ای که تعلیم وجود داری میدهیم ← $F_2(I)$ را اجرا کنیم و

جواب I بدست میاد

مسأله دوم

حالت خاص ← α_iها و β_iها روی یک الفبای unary هستند ← decidable است

مثلاً $\Sigma = \{a\}$

اگر برای یک i ای $1 \leq i \leq k$ ← $|\alpha_i| - |\beta_i| = 0$ باشد

$d_i = 0$

$= 0$ ← طول رشته بالا با پایین یکسان d_i ← جواب مسئله yes می‌شه (1)

(2) اگر برای هر iای ، $d_i > 0$ باشد یا برای هر iای $d_i < 0$ بود ← جواب مسئله No می‌شود

(3) اگر بعضی از d_iها مثبت و بعضی منفی ← جواب مسئله yes

$\left\{ \frac{a^5}{a^2} , \frac{a^4}{a^7} \right\}$ ←

از $\frac{a^5}{a^2}$ به اندازه‌ی اختلاف $\frac{a^4}{a^7}$ ← اختلاف ۳ e

$d_i = -3$ ← $d_i > 0$

یعنی ۳ تا باید داشته باشیم

و از دومی هم به اندازه‌ی اختلاف اولی ← $\cancel{3}$ ۲ ←

$\left\{ \frac{a^5}{a^2} \times 3 , \frac{a^4}{a^7} \times 2 \right\}$ ← طول رشته بالا e $15 + 2\times4$ ← (۲۳)

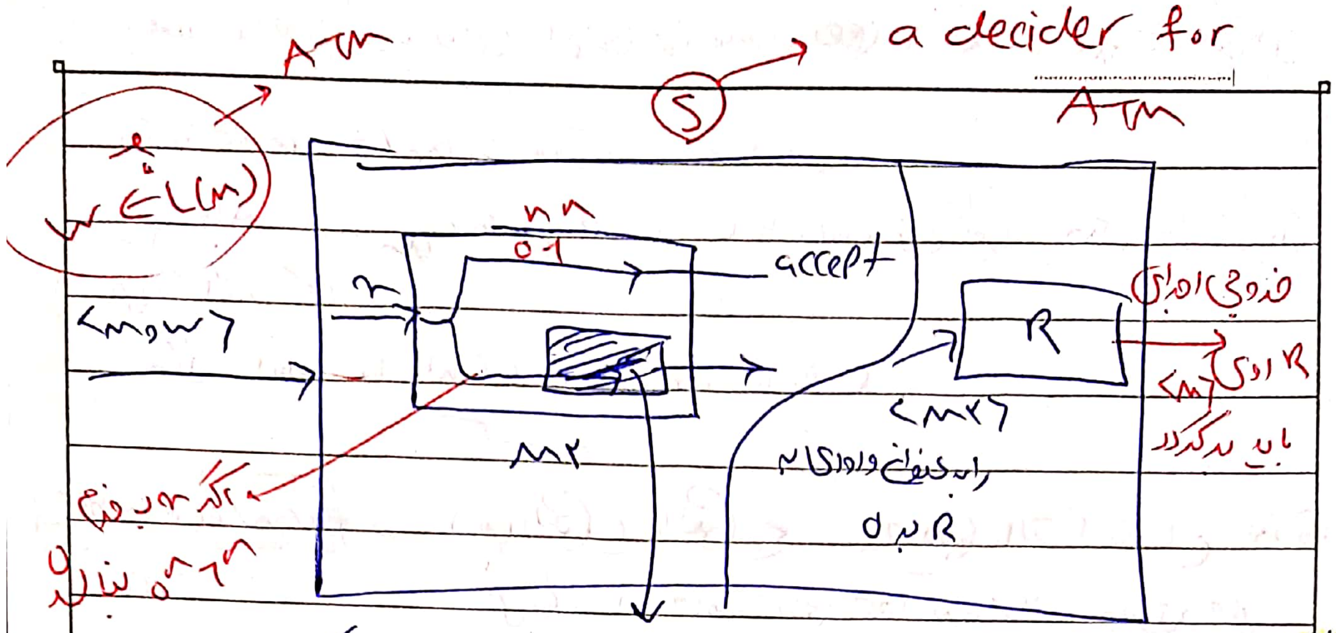
طول رشته پایین e $3\times2 + 7\times2$ = (۲۳)

undecidable ← (Regular_TM)

آیا زبان تعریف یک TM منظم است یا نه e

$REGULAR_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$

فرض خلف e مسئله حل‌پذیر است ← یک decider مثل R برای او وجود دارد ← با بهره‌گیری از R می‌توانیم decider ای مثل S برای A_{TM} طراحی کرد که با تعریف ناپذیری A_{TM} در تناقض است.



M را روی w اجرا کن و حاصل را برگردان

$w \in L(M) \Rightarrow L(M_2) = \Sigma^*$

$w \notin L(M) \Rightarrow L(M_2) = \{0^n1^n \mid n \geq 0\}$

nonregular

اگر فرض کنیم $Regular_{TM}$ decidable باشد، ATM هم decidable می‌شه تناقض

S = یک decider برای ATM ورودی $\langle M, w\rangle$ که M یک TM
و w یک string

1) یک TM به نام M2 را بساز:
M2 روی ورودی x:

1) اگر x فرم 0^n1^n داره ← accept

2) ~ ندارد ← run M on input w and accept if M accepts w

$w \in L(M) \to$ منظم $L(M_2)$

2) R را روی $\langle M_2\rangle$ اجرا کن

3) اگر R، accept کرد (فرم Σ^* الف) ← accept کن

~ reject ~ نکرد (فرم 0^n1^n الف) ~

$w \notin L(M) \to$ نامنظم $L(M_2)$

قضیه Rice: خاصیت زبان تشخیص TM ها X

اگر P یک خاصیت (nontrivial) باشد برای زبان تشخیص TM ← مسئله تعیین اینکه آیا زبان خاصیت P را دارا است undecidable است

مثل خاصیت regular بودن

* بعضی از زبان های TM ها این خاصیت را داشته باشند بعضی نداشته باشند *

مثلاً تهی بودن ← یعنی زبان ها تهی هستند یا نه ← E_{TM} مثالش این

قضیه rice undecidable است.

All_{TM} و $Regular_{TM}$ و CFL بودن زبان تشخیص یک TM طبق Rice می‌توان اثبات کرد.

مثلاً اینکه آیا $\varepsilon \in L(M)$ هست یا نه؟

undecidable ← در مورد اینکه یا نه ε ← خاصیتی است که یک زبان اتفاقاً می‌تواند و این خاصیت برای بعضی برقرار است و برای بعضی برقرار نیست.

Ex) یک CFG به اسم G و یک regular language به اسم R داریم آیا $L(G) \subseteq R$ ؟

مسئله گفته شده decidable است یا undecidable ؟

$L(G) \cap \bar{R}$

CFL یک — Regular

اشتراک یک CFL است

یک CFG متناظر داریم

$L(G) \subseteq R \iff L(G) \cap \bar{R} = \emptyset$

اگر و تنها اگر

پس باید PDA تشخیص $L(G)$ و DFA تشخیص R را در نظر بگیریم و PDA تشخیص‌دهنده را بسازیم و از روی PDA یک CFG بسازیم و سپس CFG را به یک decider برای E_{CFG} بدیم اگر accept برگرداند یعنی زبان تشخیص $\emptyset$ است ← $L(G) \subseteq R$ ← accept

اگر reject برگرداند ← $L(G) \not\subseteq R$ ← reject decidable

session 27

ALL_{DFA} ← آیا زبان تشخیص DFA داده شده برابر Σ^* است؟ ✓

ALL_{CFG} ← 〃 〃 CFG 〃 〃 ؟ ✗

ALL_{TM} ← 〃 TM 〃 〃 ✗

$A_{\varepsilon TM}$ → given a TM is $\varepsilon \in L(T)$?

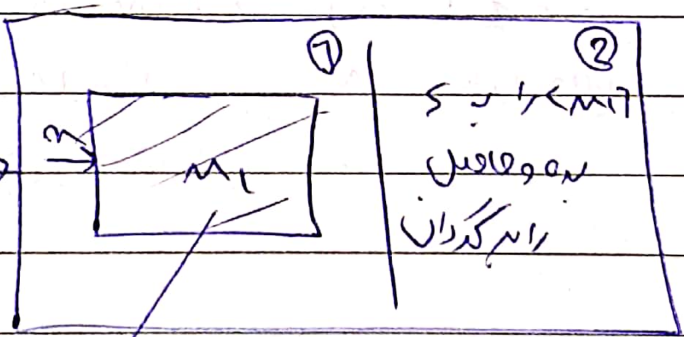
ALL_{TM} → given a TM with input alphabet Σ is $L(T) = \Sigma^*$?

فرض خلف

فرض کنیم $A_{\varepsilon TM}$ decidable باشد ← می‌توان از تصمیم‌پذیری این مسئله استفاده کرد و تصمیم‌پذیری A_{TM} را نتیجه بگیریم ← ✗ تناقض چون می‌دانیم A_{TM} undecidable است

پس یک decider مثل D برای $A_{\varepsilon TM}$ داریم و از آن می‌توان استفاده کرد و برای A_{TM} یک decider فرضی درست کنیم.

A_{TM} برای یک decider



اگر $\varepsilon = m$ باشد ← $\varepsilon \in L(M1)$ و خروجی اجرای D را برگردان

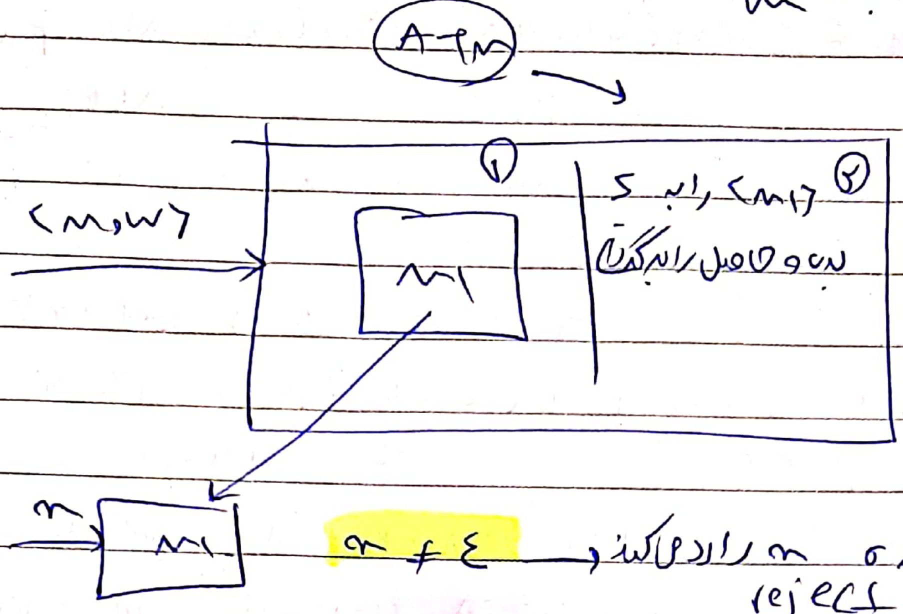
w روی نوار نوشته می‌شود با آغاز از سمت چپ‌ترین موضع و از آنجا دقیقاً مانند M رفتار می‌کند

پس $w \in L(M1) \Leftrightarrow w \in L(M)$ ← accept را برمی‌گرداند

اگر $w \notin L(M1) \Leftrightarrow w \notin L(M)$ ← reject را برمی‌گرداند

فرض کنیم ... که برای AE_{TM} وجود دارد

اثبات



$n \neq \varepsilon$ ← M_1 ، n را رد می‌کند reject

$n = \varepsilon$ ← M_1 ، حاصل اجرای M روی w را برمی‌گرداند

پس اگر $w \in L(M) \Rightarrow L(M_1) = \{\varepsilon\}$

$w \notin L(M) \Rightarrow L(M_1) = \emptyset$

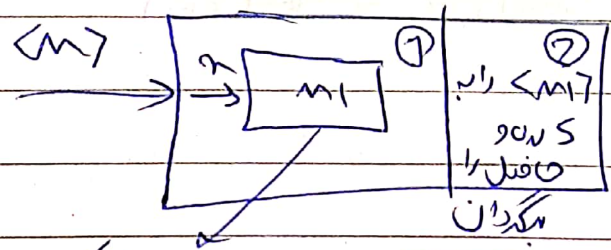
پس وجود یک decider برای AE_{TM} منجر به وجود یک decider برای A_{TM} می‌شود ← تناقض ✗

اثبات undecidable بودن ALL_{TM} به کمک AE_{TM}

فرض خلف: فرض کنیم یک decider مثل S برای ALL_{TM} وجود دارد (یعنی ALL_{TM} تصمیم پذیر است)

از کمک S یک decider برای AE_{TM} ایجاد کرد ← تناقض ✗

برای ماشین



M_1 ابتدا ورودی که روی نوار قرار دارد را پاک می‌کند و بعد از آن دقیقاً مثل M رفتار می‌کند

$\varepsilon \in L(M) \Rightarrow L(M_1) = \Sigma^*$

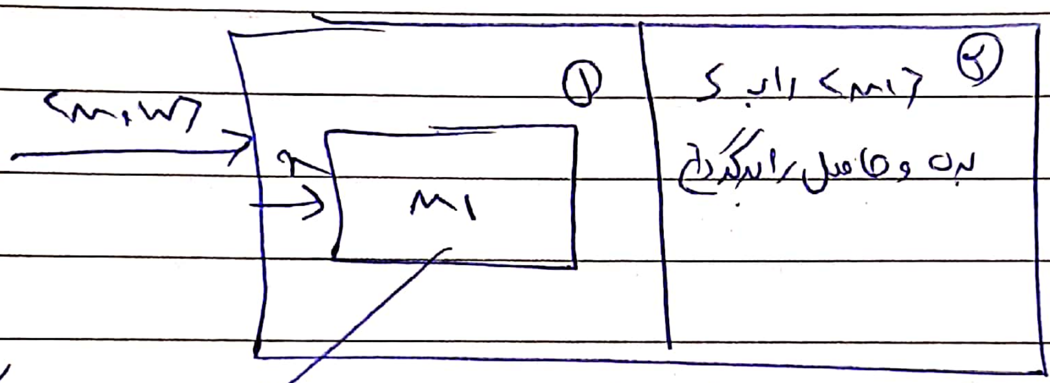
$\varepsilon \notin L(M) \Rightarrow L(M_1) = \emptyset$

اگر S و accept را برگرداند می‌فهمیم که $L(M_1)=\Sigma^*$ شده یعنی $\varepsilon \in L(M)$ پس accept می‌کنیم

اگر reject برگرداند یعنی برای $\langle M_1 \rangle$ ، $L(M_1)=\emptyset$ یعنی $\varepsilon \notin L(M)$ پس reject می‌کنیم

پس وجود یک decider مثل V برای ALL_M منجر به وجود decider برای AE_M $\Rightarrow$ ✗ تناقض

راه ۲) فرض کنید ALL_M تصمیم‌پذیر باشد (فرض خلف) و یک decider مثل R برای آن باشد با کمک R یک decider برای A_M می‌سازیم



M_1 فارغ از الفبای ورودی اش، w را به M می‌دهد و حاصل را برمی‌گرداند

پس

$w \in L(M) \Rightarrow L(M_1)=\Sigma^*$

$w \notin L(M) \Rightarrow L(M_1)=\emptyset$

اگر S و reject برگرداند یعنی $L(M_1)=\emptyset$ بوده یعنی $w \notin L(M)$

اگر S و accept برگرداند $\Leftarrow$ $L(M_1)=\Sigma^*$

$w \in L(M)$ پس تست بودن A_M را با کمک ALL_M داریم

اگر ALL_M تصمیم‌پذیر باشد، A_M هم تصمیم‌پذیر می‌شد ولی می‌دانیم نیست ✗ تناقض