اصول طراحی کامپایلر

درس‌نامه‌ی ۲

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۸
ویراست دوم: ۱۳۹۳

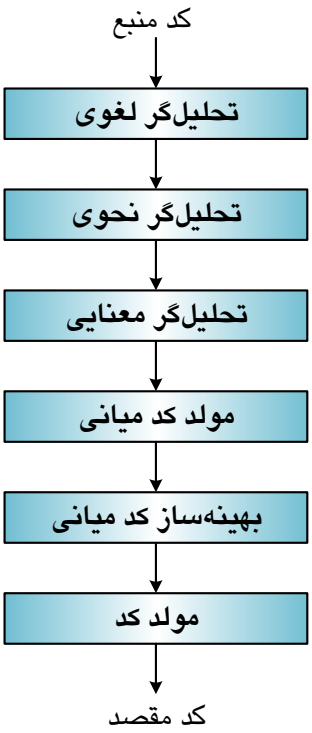
فهرست مطالب

۲ تحلیل لغوی

LEXICAL ANALYSIS

۲-۱ مقدمه

تحلیل لغوی، نخستین مرحله‌ی فرآیند کامپایل است.



شکل ۲-۱: تحلیل لغوی به عنوان نخستین مرحله‌ی فرآیند کامپایل

- در تحلیل لغوی به کد منبع به عنوان یک دنباله‌ی واحد از کاراکترها نگاه می‌شود.
- تحلیل‌گر لغوی کد منبع را می‌خواند و دنباله‌ی کاراکترها را در **توکن‌ها** گروه‌بندی می‌کند.

شکل ۲-۲: مرحله‌ی تحلیل لغوی

۲-۲ توکن، لغت و الگو

- **توکن (Token)**: کوچکترین واحد معنادار که توسط تحلیل‌گر لغوی شناسایی می‌شود.
- **لغت (Lexeme)**: دنباله‌ای از کاراکترها که یک توکن را تشکیل می‌دهد.
- **الگو (Pattern)**: قاعده‌ای که چگونگی ساخت توکن را مشخص می‌کند.

تعریف توکن و تعریف لغت به عهده‌ی تعریف زبان منبع است.

مثال

در جدول زیر نمونه‌هایی از توکن‌ها و لغت‌ها با توصیف آنها را مشاهده می‌کنید:

TOKEN	LEXEME	DESCRIPTION
const	const	language keyword
if	if	language keyword
relation	$<, <=, =, <>, >, >=$	comparison operators
id	position, A1, x	sequence of letters and digits
num	3.14, 14, 6.02E23	numeric constant

۲-۲-۱ خصیصه‌های یک توکن

خصیصه‌های یک توکن (Attributes): اطلاعات مرتبط با آن توکن

$$< Token, Attributes >$$

- از خصیصه‌های یک توکن در مرحله‌ی تحلیل نحوی و ترجمه استفاده می‌شود.
- اگر توکن مشخص کننده‌ی یک لغت واحد باشد، خصیصه‌ی آن خالی خواهد بود.

مثال

مثالی ساده از تحلیل لغوی:

کد منبع (Source): `height := 0.5 * g * t * t`

↓

لغت‌ها (Lexemes): `height` `:=` `0.5` `*` `g` `*` `t` `*` `t`

↓

توکن‌ها (Tokens): `<id,1> <:=,> <num,1> <*,> <id,2> <*,> <id,3> <*,> <id,3>`

۲-۲-۲ تمایز کلیدواژه‌ها و شناسه‌ها

- زبان‌های برنامه‌سازی از رشته‌های ثابتی برای شناسایی **کلیدواژه‌ها (Keywords)** استفاده می‌کنند، مانند: `if`، `then`، `else`، ...
- از آنجا که الگوی کلیدواژه‌ها دقیقاً همان الگوی شناسه‌هاست، تحلیل‌گر لغوی باید بین آنها تمایز قایل شود.
- اگر کلیدواژه‌ها **رزرو شده** (Reserved) باشند، می‌توانیم آنها را در ابتدای جدول نمادها قرار دهیم و آنها را با عنوان Keyword مشخص کنیم.
- در این صورت، یک رشته به عنوان یک **شناسه (Identifier)** شناسایی می‌شود، اگر در جدول نمادها به عنوان کلیدواژه علامت‌گذاری نشده باشد.

۲-۳ الفبا، رشته و زبان

- **الفبا (Alphabet)**: مجموعه‌ای متناهی و ناتهی از نمادهای تجزیه‌ناپذیر (Σ)
- **رشته (String)**: دنباله‌ای متناهی از نمادهای یک الفبا
- **زبان (Language)**: مجموعه‌ای از رشته‌های یک الفبا

۲-۳-۱ رشته‌ها

- **طول رشته**: تعداد نمادهای الفبا در رشته
- **رشته‌ی تهی**: رشته‌ای با طول صفر ϵ

عملیات روی رشته‌ها

الحاق (Concatenation):

- اگر x و y دو رشته باشند، رشته‌ی حاصل از الحاق آنها xy خواهد بود.
- ϵ عضو خنثای عمل الحاق می‌باشد:

$$x\,\epsilon = \epsilon\,x = x$$

مثال

x	y	xy
key	word	keyword
java	script	javascript

توان یک رشته:

$$\begin{aligned} x^{\circ} &= \epsilon \\ x^{i+1} &= x^i x \end{aligned}$$

ترمینولوژی اجزای رشته‌ها: پیشوند، پسوند و زیررشته

TERM	DEFINITION
prefix of s	a string obtained by removing 0 or more trailing symbols of s
suffix of s	a string obtained by removing 0 or more leading symbols of s
substring of s	a string obtained by deleting a prefix and a suffix from s
proper prefix suffix, substring of s	Any nonempty string x that is, respectively, a prefix, suffix or substring of s such that $s \neq x$

۲-۳-۲ زبان‌ها

مثال

- $\Sigma = \{0, 1\}$ – a string is an instruction
  - The set of M68K instructions
  - The set of Pentium instructions
  - The set of MIPS instructions
- $\Sigma =$ the ASCII set – a string is a program
  - the set of Haskell programs
  - the set of C programs
  - the set of VC programs

عملیات روی زبان‌ها

الحاق	Concatenation	$L_۱L_۲ = \{x_۱x_۲ : x_۱ \in L_۱, x_۲ \in L_۲\}$
توان صفر		$L^\circ = \{\epsilon\}$
توان		$L^i = L^{i-۱}L$
اجتماع	Union	$L_۱ \cup L_۲ = \{x : x \in L_۱, x \in L_۲\}$
بستار ستاره‌ای	Star Closure	$L^* = \cup_{i=۰}^{\infty} L^i$
بستار مثبت	Positive Closure	$L^+ = \cup_{i=۱}^{\infty} L^i$

$$L^* = L^+ \cup \{\epsilon\}$$

مثال

مثال‌هایی از عملیات روی زبان‌ها

- $L = \{a, \ldots, z, A, \ldots, Z, \_\}$
- $D = \{0, \ldots, 9\}$

EXAMPLE	LANGUAGE
$L \cup D$	letters and digits
LD	strings consisting of a letter followed by a digit
L^3	all 3-letter strings
L^*	all strings of letters, including the empty string ϵ
$L(L \cup D)^*$	all strings of letters and digits beginning with a letter
D^+	all strings of one or more digits

۲-۴ مؤلفه‌های تحلیل‌گر لغوی

برای یک تحلیل‌گر لغوی می‌توان سه مؤلفه‌ی زیر را در نظر گرفت:

- **زبان مشخص‌سازی تحلیل‌گر لغوی**: برای بیان الگوی توکن
- **مکانیزم تحلیل‌گر لغوی**: نحوه‌ی تطبیق لغت‌ها با الگوها
- **پیاده‌سازی تحلیل‌گر لغوی**: نحوه‌ی کدگذاری و پیاده‌سازی تحلیل‌گر لغوی

در قسمت‌های بعدی این فصل به تشریح موارد فوق می‌پردازیم.

۲-۵ عبارت‌های منظم: ابزاری برای تعریف الگوی توکن‌ها

عبارت منظم برای توصیف الگوی توکن شناسه:

`letter(letter|digit)`*

۲-۵-۱ عبارت منظم (Regular Expression)

فرض می‌کنیم که Σ یک الفبای داده شده باشد:

۱) $\varnothing$، ϵ و a $(a \in \Sigma)$ عبارت‌های منظم هستند (عبارات منظم ابتدایی).

۲) اگر r_1 و r_2 دو عبارت منظم باشند، در این صورت

- $r_1|r_2$،
- $r_1.r_2$،
- r_1^* و
- (r_1)

هم عبارات منظم هستند.

۳) یک رشته عبارت منظم است، اگر و فقط اگر بتوان آن را از ترکیب عبارات منظم ابتدایی با بکارگیری قانون ۲ به تعداد متناهی به دست آورد.

۲-۵-۲ زبان توصیف شده بایک عبارت منظم

اگر r یک عبارت منظم باشد، $L(r)$ زبان مرتبط با r است و مطابق جدول زیر تعریف می‌شود:

$L(\varnothing) = \varnothing$
$L(\epsilon) = \{\epsilon\}$
$L(a) = \{a\} \quad , a \in \Sigma$
$L(r_1\|r_2) = L(r_1) \cup L(r_2)$
$L(r_1.r_2) = L(r_1).L(r_2)$
$L((r)) = L(r)$
$L(r^*) = (L(r))^*$

مثال

نمونه‌هایی از عبارت‌های منظم:

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

Regular Expression	Language
$a \mid b$	$\{a, b\}$
$(a \mid b)(a \mid b)$	$\{aa, ab, ba, bb\}$
a^*	$\{\epsilon, a, aa, aaa, \ldots\}$
$(a \mid b)^*$	Σ^*

مثال

نمونه‌ای از محاسبه‌ی زبان یک عبارت منظم:

- Alphabet: $\Sigma = \{0, 1\}$
- RE: $0(0|1)^*$
- Question: What is the language defined by the RE?
- Answer:

$$\begin{aligned} L(0(0|1)^*) &= L(0)L((0|1)^*) \\ &= \{0\}L(0|1)^* \\ &= \{0\}(L(0) \cup L(1))^* \\ &= \{0\}(\{0\} \cup \{1\})^* \\ &= \{0\}\{0, 1\}^* \\ &= \{0\}\{\epsilon, 0, 1, 00, 01, 10, 11, \dots\} \\ &= \{0, 00, 01, 000, 001, 010, 011, \dots\} \end{aligned}$$

The RE describes the set of strings of 0's and 1's beginning with a 0.

مثال

چند عبارت منظم و زبان متناظر با آنها:

RE	LANGUAGE
1	$\{1\}$
$0\|1$	$\{0, 1\}$
1^*	$\{\epsilon, 1, 11, 111, \dots\}$
1^*1	$\{1, 11, 111, \dots\}$
$0\|0^*1$	the set containing 0 and all strings consisting of zero or more 0's followed by a 1.

۲-۵-۳ کوته‌نوشت‌هایی برای عبارت‌های منظم

بستار مثبت

$$r^+ = r.r^*$$

نماد وجود

$$r^? = r \mid \epsilon$$

طبقه‌های کاراکتر (character class):

$$[۱, ۳, ۵] = ۱ \mid ۳ \mid ۵$$

$$[a - z] = a \mid b \mid \cdots \mid z$$

۲-۵-۴ تقدم عملگرها برای عبارت‌های منظم

برای اجتناب از پرانتزگذاری‌های زیاد، از تقدم عملگرها استفاده می‌کنیم

بستار ستاره‌ای	*
الحاق	.
اجتماع	|

کاهش تقدم ↓

بالاترین تقدم مربوط به بستار ستاره‌ای و پس از آن به الحاق و سپس اجتماع می‌باشد.

۲-۵-۵ تعریف‌های منظم (Regular Definitions)

از تعریف‌های منظم برای نامگذاری عبارت‌های منظم استفاده می‌کنیم:

عبارت منظم → نام عبارت منظم

در یک تعریف منظم، می‌توان دنباله‌ای از تعاریف فوق را استفاده نمود و نام هر یک از عبارت‌های منظم را در تعریف دیگری به کار برد.

مثال

تعریف منظم برای شناسه:

$$\begin{aligned} letter &\rightarrow A \mid B \mid \cdots \mid Z \mid a \mid b \mid \cdots \mid z \\ digit &\rightarrow ۰ \mid ۱ \mid \cdots \mid ۹ \\ id &\rightarrow letter(letter \mid digit)^* \end{aligned}$$

به طور معادل (با استفاده از طبقه کاراکترها)

$$id \rightarrow [A-Za-z][A-Za-z۰-۹]^*$$

مثال

تعریف منظم برای اعداد:

$$5230, \quad 3.14, \quad 6.45E4, \quad 1.84E\text{-}4$$

$$\begin{aligned} digit &\rightarrow ۰ \mid ۱ \mid \cdots \mid ۹ \\ sign &\rightarrow + \mid - \mid \epsilon \\ digits &\rightarrow digit^+ \\ fraction &\rightarrow (.digits)^? \\ exponent &\rightarrow (E\ sign^?\ digits)^? \\ num &\rightarrow sign\ digits\ fraction\ exponent \end{aligned}$$

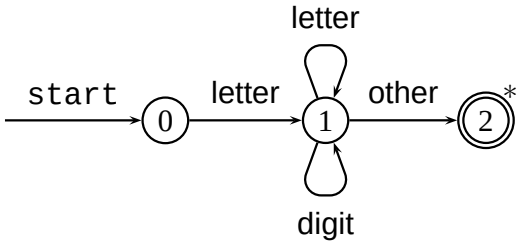
۲-۵-۶ گرامرهای منظم به جای عبارت‌های منظم

گرامرهای منظم یا گرامرهای نوع سه

- هر عبارت منظم با یک گرامر منظم متناظر است و برعکس
- عبارت‌های منظم نسبت به گرامرهای منظم شکل فشرده‌تری دارند و خواندن آنها ساده‌تر است.

۲-۶ آتوماتای متناهی: مکانیزمی برای بازشناسی توکن‌ها

آتوماتون متناهی برای بازشناسی توکن شناسه:



۲-۶-۱ آتوماتون متناهی قطعی (DFA)

آتوماتون متناهی حالت قطعی Deterministic Finite Automata (DFA)، یک پنج‌تایی مرتب به صورت

$$M = (Q, \Sigma, \delta, q_\circ, F)$$

است که در آن

Q **مجموعه‌ی حالات**: مجموعه‌ی متناهی و ناتهی از حالت‌های داخلی *(internal states)*

Σ **مجموعه‌ی الفبای ورودی** *(alphabet)*

δ **تابع گذر حالت** به صورت $\delta : Q \times \Sigma \to Q$ که حالت بعدی آتوماتون را بر اساس حالت فعلی و نماد ورودی جاری تعیین می‌کند *(state transition function)*

$q_\circ$ **حالت آغازین** $q_\circ \in Q$ *(initial state)*

F **مجموعه‌ی حالات نهایی** $F \subseteq Q$ *(final states)*

تابع گذر حالت تعمیم‌یافته برای آتوماتون متناهی قطعی

$$\begin{aligned} \delta^* &: Q \times \Sigma^* \to Q \\ \delta^*(q, \epsilon) &= q \\ \delta^*(q, xa) &= \delta(\delta^*(q, x), a), \qquad x \in \Sigma^*, a \in \Sigma \end{aligned}$$

رشته‌های پذیرفته شده توسط یک DFA داده شده

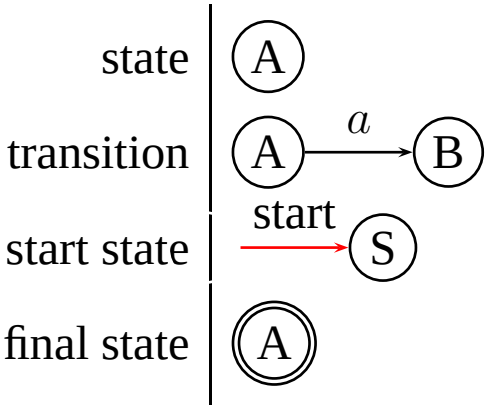
DFA رشته‌ی ورودی x را می‌پذیرد، اگر با شروع از حالت آغازین پس از مصرف x در یکی از حالت‌های نهایی توقف کند.

$$\delta^*(q_\circ, x) = q_f, \qquad q_f \in F$$

گراف گذر حالت برای بازنمایی آتوماتون متناهی

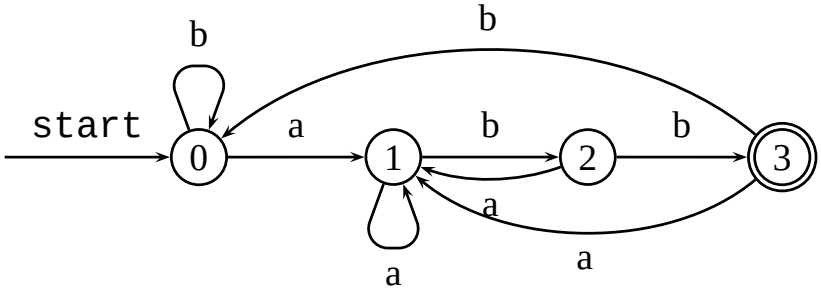
گراف گذر حالت:

- **گره‌ها:** حالت‌های آتوماتون
  - حالت آغازین: با یک پیکان ورودی به آن
  - حالت‌های نهایی: با دو دایره
- **یال‌ها:** تابع گذر میان حالت‌ها



مثال

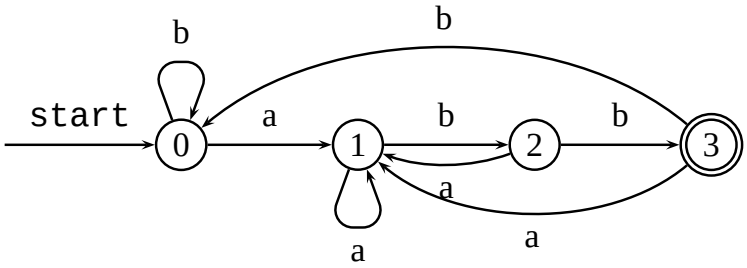
نمونه‌ای از گراف گذر حالت برای بازنمایی آتوماتون متناهی: یک DFA برای پذیرش رشته‌های زبان $L((a \mid b)^*abb)$



جدول گذر حالت برای بازنمایی آتوماتون متناهی

تابع گذر حالت را با جدول نیز می‌توان نمایش داد.

مثال



q	$\delta(q,a)$	$\delta(q,b)$
0	1	0
1	1	2
2	1	3
3	1	0

۲-۶-۲ آتوماتون متناهی غیر قطعی (NFA)

آتوماتون متناهی حالت غیرقطعی Non-deterministic Finite Automata (NFA)، یک پنج‌تایی مرتب به صورت

$$M = (Q, \Sigma, \delta, q_\circ, F)$$

است که در آن

Q **مجموعه‌ی حالات**: مجموعه‌ی متناهی و ناتهی از حالت‌های داخلی *(internal states)*

Σ **مجموعه‌ی الفبای ورودی** *(alphabet)*

δ **تابع گذر حالت** به صورت $\boxed{\delta : Q \times (\Sigma \cup \{\epsilon\}) \to ۲^Q}$ که حالت بعدی آتوماتون را بر اساس حالت فعلی و نماد ورودی جاری تعیین می‌کند *(state transition function)*

$q_\circ$ **حالت اولیه** $q_\circ \in Q$ *(initial state)*

F **مجموعه‌ی حالات نهایی** $F \subseteq Q$ *(final states)*

تابع گذر حالت تعمیم‌یافته برای آتوماتون متناهی غیرقطعی

$$\begin{aligned} \delta^* &: Q \times \Sigma^* \to ۲^Q \\ \delta^*(q,\epsilon) &= \{q\} \\ \delta^*(q,xa) &= \cup_{q_i \in \delta^*(q,x)} \delta(q_i,a), \qquad x \in \Sigma^*, a \in \Sigma \end{aligned}$$

با داشتن یک رشته و یک NFA، در حالت کلی بیش از یک مسیر می‌تواند برای پیمودن وجود داشته باشد.

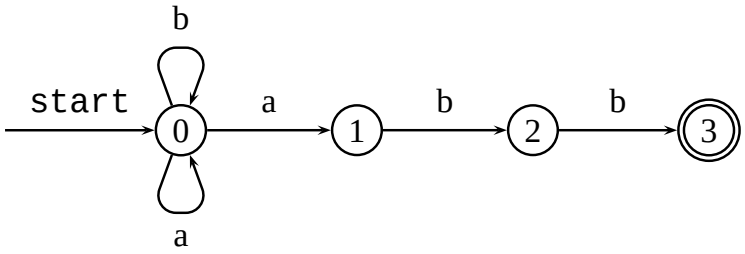
رشته‌های پذیرفته شده توسط یک NFA داده شده

NFA رشته‌ی ورودی x را می‌پذیرد، اگر با شروع از حالت آغازین حداقل یک مسیر منتهی به یک حالت نهایی وجود داشته باشد:

$$\exists q_f,\ q_f \in \delta^*(q_\circ, x), \quad q_f \in F$$

مثال

یک NFA برای پذیرش رشته‌های زبان $L((a \mid b)^*abb)$



مثال: پذیرش رشته‌ی $aaabb$

مقایسه‌ی آتوماتون‌های قطعی و غیرقطعی

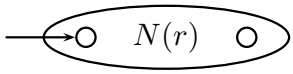
- هم DFA و هم NFA قابلیت پذیرش همه‌ی عبارت‌های منظم را دارند.
- معمولاً اجرای DFA روی یک رشته سریع‌تر از اجرای NFA روی همان رشته است.
  (عدم وجود Backtracking در پیاده‌سازی DFA روی ماشین‌های ترتیبی)
- تعداد حالت‌های DFA معمولاً به طور نمایی بیشتر از NFA است.

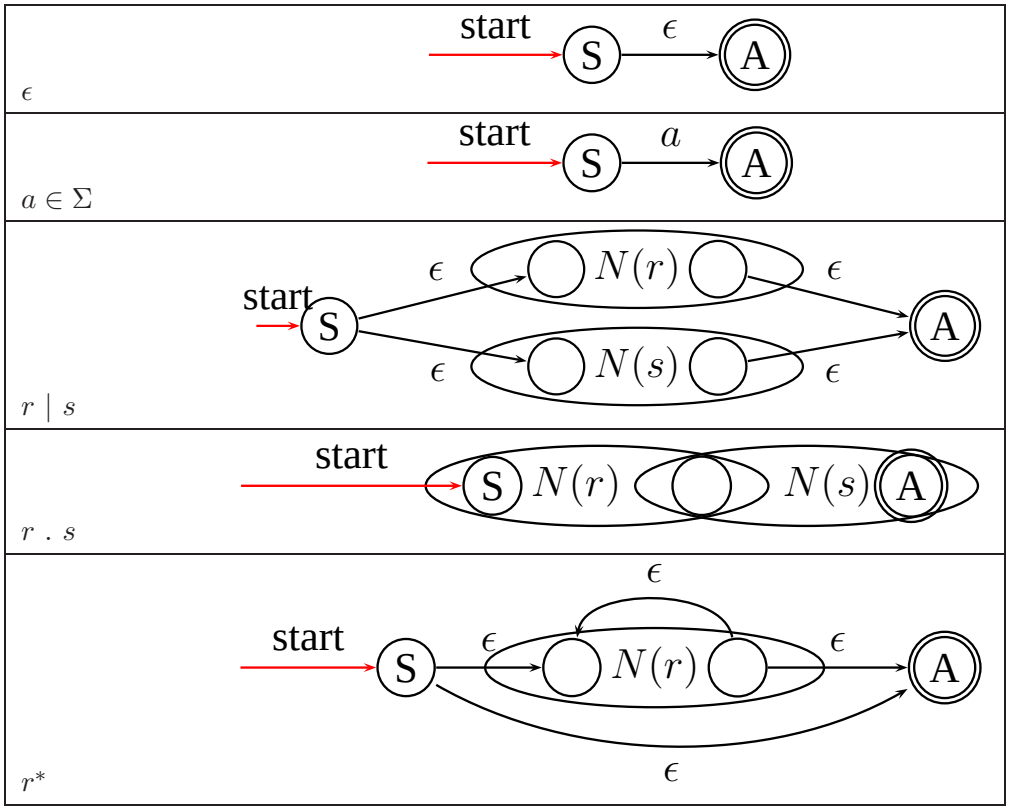
۳-۶-۲ ساخت NFA متناظر با یک عبارت منظم

Thompson's Construction برای ساخت NFAی پذیرنده‌ی یک عبارت منظم:
NFAی حاصل:

- دقیقاً یک حالت نهایی دارد.
- هیچ یالی به حالت آغازین آن وارد نمی‌شود.
- هیچ یالی از حالت نهایی خارج نمی‌شود.

اگر r یک عبارت منظم باشد، $N(r)$ پذیرنده‌ی متناهی غیرقطعی متناظر با آن خواهد بود.

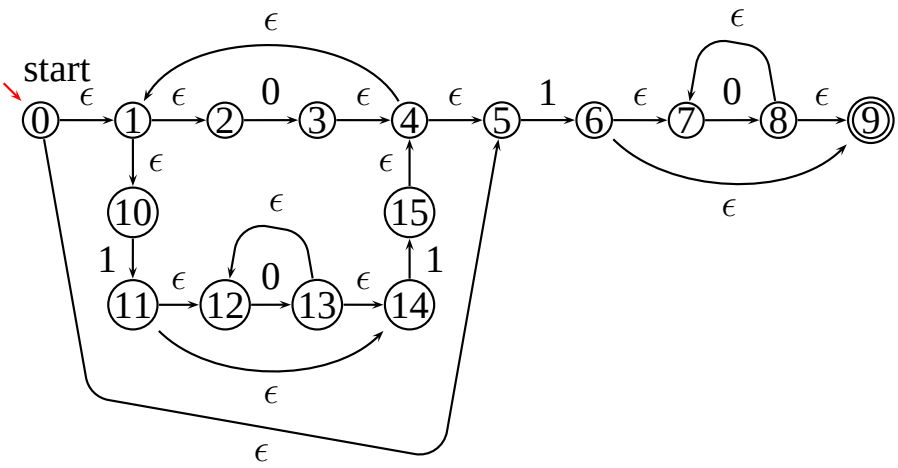




مثال

نمونه‌ی ساخت NFA متناظر با یک عبارت منظم:

$$(0 \mid 10^*1)^*10^*$$



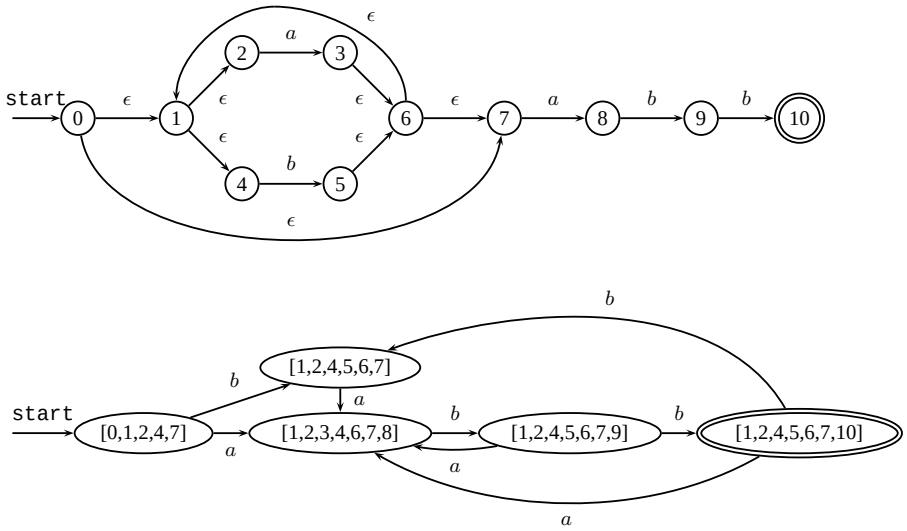
۲-۶-۴ ساخت یک DFA متناظر با یک NFA داده شده

Subset Construction

- حالت‌های DFA را به عنوان زیرمجموعه‌های حالت‌های NFA در نظر می‌گیریم.
- حالت شروع، حالت $\delta^*(q_\circ, \epsilon)$ خواهد بود.
- حالت‌های نهایی DFA حالت‌هایی هستند که در آنها حداقل یکی از حالت‌های نهایی NFA وجود داشته باشد.
- تابع گذر حالت DFA برای هر حالت و هر نماد الفبا، اجتماع مقادیر تابع گذر NFA برای حالت‌های موجود در آن حالت DFA با آن نماد الفباست.

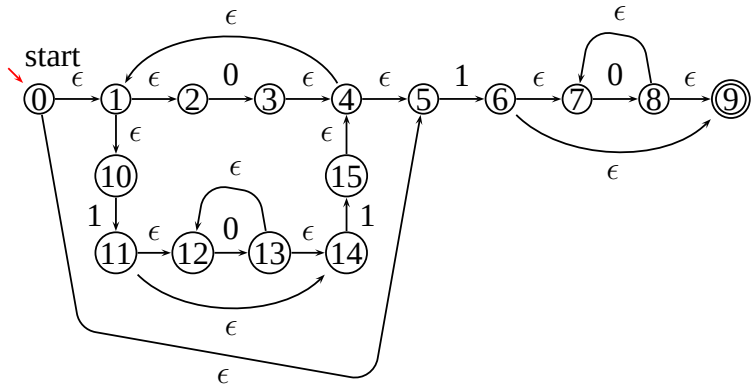
مثال

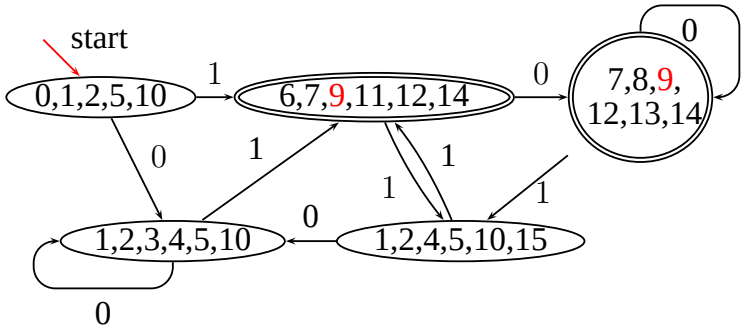
نمونه‌ی ساخت یک DFA متناظر با یک NFA



مثال

نمونه‌ای دیگر از ساخت یک DFA متناظر با یک NFA

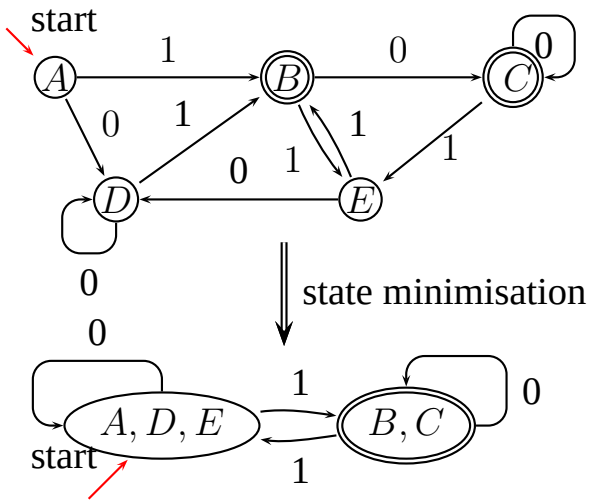




۲-۶-۵ می‌نیمم سازی تعداد حالات یک DFA داده شده

- برای هر DFA یک DFAی معادل با حداقل تعداد حالات وجود دارد.

مثال



۲-۷ پیاده‌سازی تحلیل‌گر لغوی

- **تحلیل‌گر لغوی باید قادر به انجام دو چیز باشد:**
  ۱) بازشناسی زیررشته‌های متناظر با لغت‌ها
  ۲) برگرداندن زوج $< Token, Attribute >$ برای هر لغت
- **در مقایسه‌ی این روال با آتوماتا**
  ۱) آتوماتا یک رشته را می‌پذیرد یا رد می‌کند، اما آن را بخش‌بندی نمی‌کند؛

۲) ⇐ بنابراین باید کاری بیشتر از پیاده‌سازی یک آتوماتون ساده انجام شود.

- **دو پیامد مهم:**

۱) هدف بخش‌بندی برنامه‌ی منبع است:
این کار با خواندن کد از چپ به راست و بازشناسی یک توکن در هر مرحله انجام می‌شود.

۲) برخورد با **ابهام‌ها**: برای مثال، `pos` یا `position` به عنوان شناسه؟

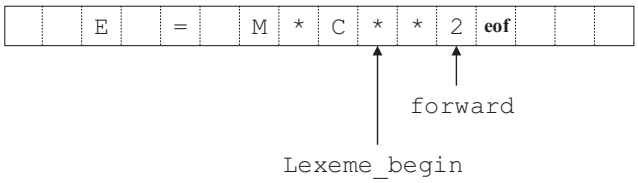
۲-۷-۱ خواندن ورودی

- تحلیل لغوی، تنها مرحله از کامپایل است که باید ورودی را بخواند.
- خواندن ورودی می‌تواند زمان‌گیر باشد. برای بهبود کارایی، ضروری است که تکنیک‌های بافربندی مناسب مورد استفاده قرار گیرد.
- از دو اشاره‌گر به بافر ورودی استفاده می‌کنیم:

۱) در ابتدا هر دو اشاره‌گر به نخستین کاراکتر لغتی که باید پیدا شود، اشاره می‌کنند.

۲) اشاره‌گر `forward` به جلو حرکت می‌کند تا اینکه لغت بازشناسی شود.

۳) اگر این لغت به طور موفق پردازش شد، هر دو اشاره‌گر به کاراکتر بلافاصله بعد از آن لغت اشاره خواهند کرد.



۲-۷-۲ از عبارت‌های منظم تا تحلیل‌گر لغوی

۱) برای الگوی هر توکن، عبارت منظم متناظر با آن را می‌نویسیم.

۲) برای هر عبارت منظم، آتوماتون متناظر با آن را می‌سازیم.

۳) ورودی را می‌خوانیم و با آزمودن آتوماتون‌های مختلف، لغت‌ها را با توکن‌ها متناظر می‌سازیم:

- اگر با دنبال کردن یک آتوماتون شکست خوردیم، اشاره‌گر `forward` را به عقب برمی‌گردانیم (با استفاده از اشاره‌گر دوم) و آتوماتون بعدی را آزمایش می‌کنیم؛
- اگر همه‌ی آتوماتون‌ها شکست خوردند، در این صورت یک **خطای لغوی** آشکار می‌شود؛
- در هنگام بازشناسی یک لغت، یک کنش (action) می‌تواند اجرا شود (مانند درج کردن رکورد یک شناسه در جدول نمادها و ...).

برخورد با ابهام‌ها

در تحلیل لغوی، دو دسته ابهام ممکن است:

۱) یک لغت واحد توسط دو یا چند عبارت منظم مختلف بازشناسی شود.

۲) یک عبارت منظم واحد بتواند بخشی از یک لغت را نیز بازشناسی کند.

برخورد با ابهام‌ها (۱) دو عبارت منظم زیر را در نظر بگیرید:

$$R_1 \rightarrow abb$$
$$id \rightarrow letter(letter \mid digit)^*$$

لغت abb هم با R_1 و هم با id تطابق می‌یابد.

> **قاعده:**
> اگر یک لغت توسط چند عبارت منظم بازشناسی می‌شود، عبارت منظمی که در لیست جلوتر قرار گرفته است، در نظر گرفته می‌شود.

برخورد با ابهام‌ها (۲) عبارت منظم مربوط به شناسه‌ها و رشته‌ی `position` را در نظر بگیرید:

$$id \rightarrow letter(letter \mid digit)^*$$

لغت‌های زیر، همگی با id تطابق می‌یابند:

- `p`
- `po`
- `...`
- `position`

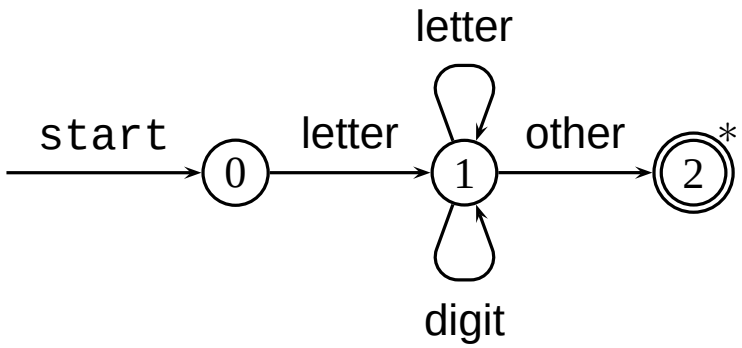
> **قاعده: (استراتژی حداکثر طول)**
> لغت مربوط به یک توکن آن است که بیشترین طول ممکن را دارد.

برای پیاده‌سازی استراتژی حداکثر طول، دو راه وجود دارد:

۱) پیاده‌سازی اول: استفاده از آتوماتا با Lookahead (نماد پیش‌بینی)

مثال

آتوماتون شناسه‌ها



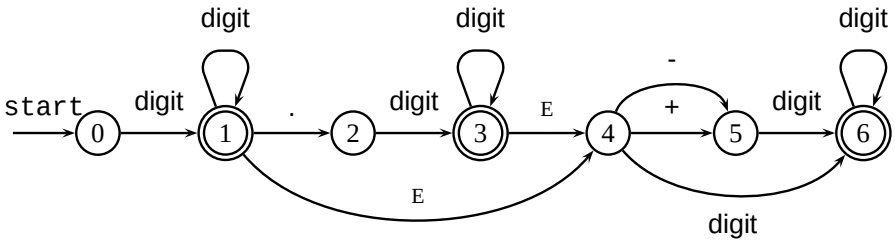
- منظور از برچسب `other` هر کاراکتری است که یال‌های ترک‌کننده‌ی آن گره با آن علامت‌گذاری نشده باشد.
- منظور از * این است که یک کاراکتر اضافی خوانده شده است و باید اشاره‌گر `forward` را یک واحد به عقب برگردانیم.

۲) پیاده‌سازی دوم: تغییر پاسخ آتوماتون به عدم پذیرش

- با رسیدن به حالت پذیرش، توقف نمی‌کنیم؛
- وقتی به شکست برخوردیم، به آخرین حالت پذیرش برمی‌گردیم.

مثال

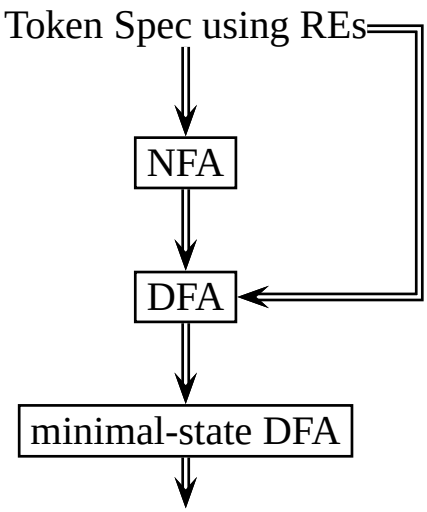
آتوماتون اعداد (61.23Express)



نکته

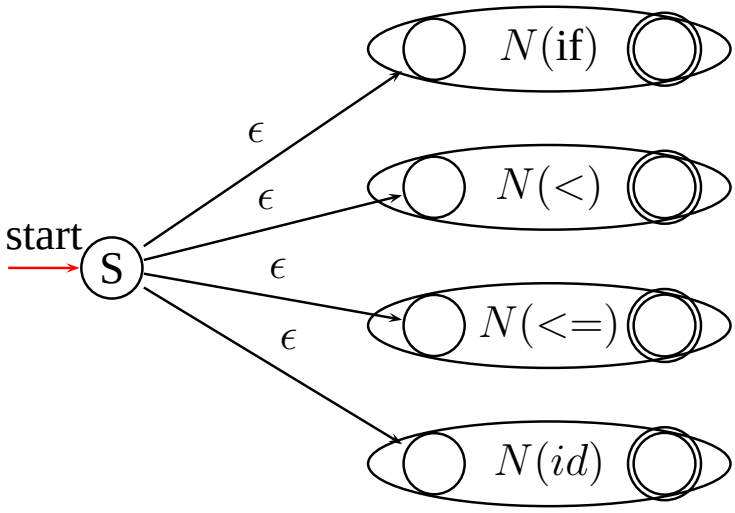
ترتیب آتوماتون‌ها اهمیت دارد (به خاطر بیاورید که آتوماتون‌ها یکی پس از دیگری در یک دنباله آزمایش می‌شوند): اولین آتوماتون‌ها را آنهایی قرار دهید که پر رخدادترین توکن‌ها هستند (مانند فواصل خالی).

۲-۷-۳ فرآیند ساخت بازشناس توکن‌ها

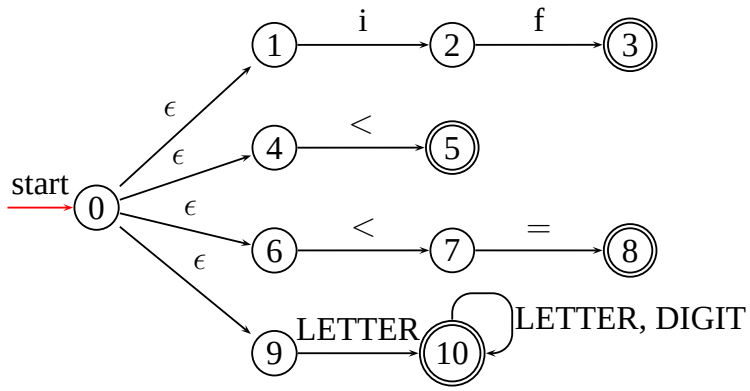


مثال

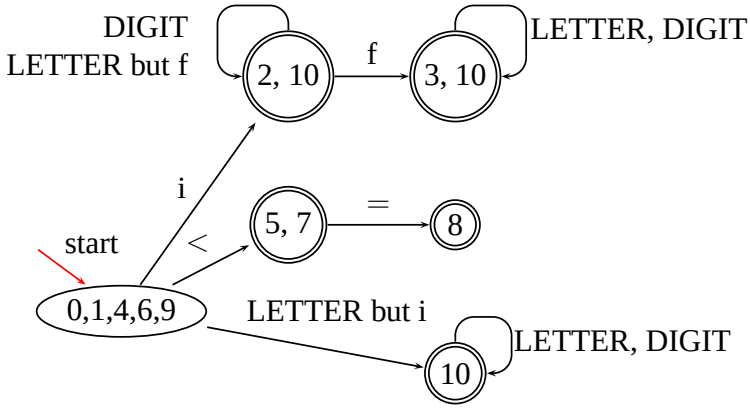
NFA برای بازشناسی چهار توکن if، <، =< و id



تکمیل NFA



تبدیل NFA به DFA و می‌نیمم‌سازی DFA



۲-۸ مسایل خاص برای تحلیل‌گر لغوی

تحلیل‌گر لغوی برای تشخیص یک توکن، گاهی مجبور می‌شود که چند کاراکتر اضافی را نیز از ورودی بخواند. کاراکترهای اضافی خوانده شده پس از تشخیص توکن باید به بافر ورودی برگردانده شوند. در قسمت‌های زیر به مواردی چند از این موضوع اشاره می‌کنیم.

۲-۸-۱ کلمات کلیدی

در برخی زبان‌های برنامه‌سازی، کلمات کلیدی رزرو شده نیستند (مانند PL/1).

مثال

جملات زیر در زبان PL/1 معتبر هستند. در اینجا کلمات کلیدی مربوط به ساختار شرطی برای متغیرها نیز استفاده شده‌اند!

```
if then then then = else;
else else = then;
```

۲-۸-۲ فاصله‌های خالی بااهمیت

در برخی زبان‌های برنامه‌سازی، فواصل خالی نادیده گرفته می‌شوند (مانند FORTRAN و Algol68).

مثال

در دو جمله‌ی زیر از زبان FORTRAN تا زمانی که به کاراکتر کاما نرسیم مشخص نیست که این دستور یک حلقه را مشخص می‌کند که برای متغیر `i` از 1 تا 25 اجرا می‌شود یا انتساب مقدار 1.25 به متغیر `do10i` می‌باشد!

```
do 10 i = 1,25
do 10 i = 1.25
```

۲-۸-۳ ثوابت رشته‌ای

کاراکترهای خاص در ثابت‌های رشته‌ای باید به گونه‌ای دیگر تفسیر شوند، مانند:

- کاراکتر خط جدید
- `tab`
- علامت نقل قول ’’ ‘‘
- جداکننده‌های توضیحات (comment delimiter)

۲-۸-۴ بستار متناهی

در برخی زبان‌های برنامه‌سازی، طول شناسه‌ها محدودیت دارد؛ بنابراین، علاوه بر الگوی توکن که با آتوماتون متناهی بازشناسی می‌شود، باید از مکانیزم مکملی برای تشخیص طول لغت نیز استفاده کرد.
(مانند FORTRAN 66: حداکثر ۶ کاراکتر برای هر شناسه)

۲-۹ خطاهای لغوی

خطای لغوی، خطایی است که در سطح تحلیل لغوی شناسایی می‌شود.

- هرگاه یک لغت با الگوی هیچ توکنی مطابقت نکند، خطای لغوی رخ داده است.
- تعداد کمی از خطاها را می‌توان در سطح لغوی تشخیص داد (به دلیل دید محلی).

۲-۹-۱ عبور از خطاهای لغوی: استراتژی حالت وحشت (Panic Mode)

Panic Mode یک استراتژی ساده برای عبور از خطای لغوی:

از ورودی باقیمانده کاراکترها را به ترتیب حذف می‌کنیم تا زمانی که تحلیل‌گر قادر به یافتن یک توکن جدید شود.

۲-۹-۲ عبور از خطاهای لغوی: استراتژی‌های تک کاراکتری

در این استراتژی‌ها، سعی می‌کنیم با درج، حذف یا تغییر یک کاراکتر، از خطا عبور کنیم.

حذف یک کاراکتر بیگانه	تبدیل `=$:` به `:=`
درج یک کاراکتر جا افتاده	تبدیل `:` به `:=`
جایگزینی یک کاراکتر ناصحیح با کاراکتر صحیح	تبدیل `::` به `:=`
تغییر مکان دو کاراکتر همسایه	تبدیل `:=` به `=:` یا تبدیل `fi` به `if`

۲-۹-۳ عبور از خطاهای لغوی: استراتژی کم‌ترین فاصله

یک راه برای عبور از خطا در برنامه، محاسبه‌ی حداقل تعداد تغییر شکل رفع خطا برای تبدیل برنامه‌ی خطا به یک برنامه‌ی صحیح است.

رفع خطا با استراتژی کم‌ترین فاصله، به عنوان یک قاعده به منظور داشتن معیاری نظری است و اغلب پیاده‌سازی نمی‌شود؛ زیرا هزینه‌ی پیاده‌سازی آن بالاست.

◄ تمرین

۱. زبانی که توسط هر یک از عبارت‌های منظم زیر توصیف می‌شود را مشخص کنید.

آ) $۰(۰|۱)^*۰$

ب) $((\epsilon|۰)۱^*)^*$

پ) $(۰|۱)^*۰(۰|۱)(۰|۱)^*$

ت) $^*۰۱۰^*۱۰^*۰$

۲. برای هر یک از مجموعه رشته‌های زیر یک تعریف منظم بنویسید.

آ) تمامی رشته‌های متشکل از حروف زبان انگلیسی که حروف آنها به ترتیب الفبایی قرارگرفته باشند.

ب) تمامی رشته‌های اعداد که دارای رقم تکراری نیستند.

پ) تمامی رشته‌های متشکل از ۰ و ۱ که حاوی زیررشته‌ی ۰۱۱ نیستند.

ت) تمامی رشته‌های متشکل از ۰ و ۱ که شامل تعداد زوجی ۰ و تعداد فردی ۱ باشد.

۳. برای مجموعه رشته‌های شامل یک b و حداقل سه a روی الفبای $\Sigma = \{a, b\}$،

آ) یک عبارت منظم بنویسید.

ب) از روی عبارت منظم، NFAی پذیرنده‌ی آن را رسم کنید.

پ) یک DFA با حداقل تعداد حالت‌ها معادل با NFAی قسمت قبل ایجاد کنید.

۴. یک DFA برای بازشناسی توکن‌های زیر طراحی کنید:

$$= -, = *, = +, =, ==, =>, <=, <$$

۵. یک آتوماتون متناهی برای پذیرش آدرس‌های ایمیل یا آدرس‌های وب طراحی کنید. دقت کنید که دامنه‌ها می‌توانند مرتبه‌ی دوم (مثل `.ac.ir`) و یا مرتبه بالاتر (مثل `.ut.ac.ir`) باشد. (راهنمایی: از استراتژی حداکثر طول استفاده کنید).

۶. توضیح دهید که چرا برای زبان‌های زیر، نمی‌توان یک عبارت منظم برای توصیف الگوی رشته‌ها پیدا کرد:

آ) رشته‌های متشکل از a و b که تعداد aهای آن بیشتر از تعداد bها باشد.

ب) رشته‌های متشکل از a و b که متقارن هستند.

۷. برای زبان‌های زیر عبارت منظم بنویسید:

آ) تمامی اعداد دودویی که مضرب ۵ هستند.

ب) تمامی اعداد دودویی که بزرگ‌تر از ۱۰۱۰۰۱ هستند.

۸. رشته زیر را درنظر بگیرید:

$$AcaacbBbAabaAAabbaAacBbABBBBc$$

که تحلیل‌گر لغوی به ازای آن لغت‌های زیر را شناسایی کرده است:

$$Ac \quad aacbBb \quad AabaA \quad AabbaA \quad acBb \quad ABBBBc$$

عبارت‌های منظمی را برای تحلیل‌گر لغوی بنویسید که قادر به شناسایی دنباله‌ی لغت‌های بالا برای رشته ورودی ذکر شده باشد .عبارت‌های منظم نوشته شده فقط باید شامل عملگرهای بستار و الحاق باشند.

۹. خطاهای لغوی موجود در تکه کد زیر که به زبان C نوشته شده است را مشخص کنید و چگونگی برخورد با هر یک از آنها را بیان کنید:

```
1 @include <iostream>
2
3 int main(int argc, char* argv[]){
4   int a;
5   if(argc =+ 1))
6     a = 12.01;
7   else argv[1] = 'salam';
8
9   int 12num = 3;
10   return1;
11  /*What?
12 }
```