Subject : طراحی کامپایلر

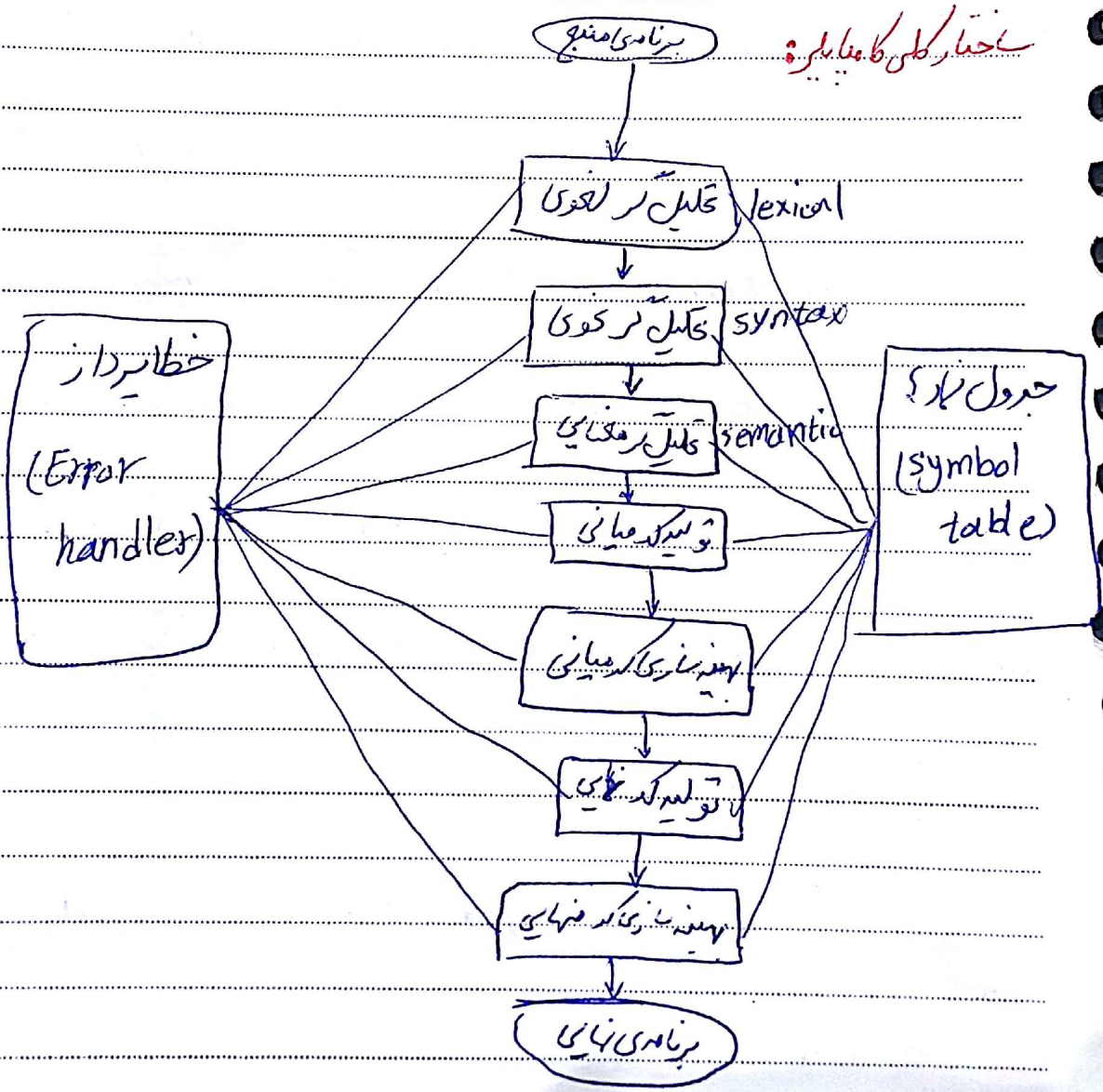
ate .....

مرجع: کتاب طراحی و ساخت کامپایلر؛ مولف: فردین شاپوری، انتشارات پوران پژوهش

میان‌ترم: 25٪ تمرین: 10٪ پایان‌ترم: 65٪

تعریف کامپایلر: برنامه‌ای است که یک برنامه‌ی منبع (source program) را به عنوان ورودی دریافته و آن را به یک برنامه‌ی پایی (target program) تبدیل می‌کند.

ساختار کلی کامپایلر:



Subject : Date

تحلیل لغوی: برنامه‌ی منبع از نظر لغوی (واژه‌ای) مورد بررسی و تحلیل قرار می‌گیرد.

نحوه‌ی برخورد و رفع خطا { - نادیده گرفتن و رد شدن تا نقطه‌ای امن
- اصلاح خطا در محدوده‌ی وقوع خطا

* انواع واژه‌های (توکن) متداول زبان‌های برنامه‌نویسی:

- شناسه‌ها id=100 - اعداد - جداکننده‌ها {};() - کلمات کلیدی while, for, if, ...

- عملگرها - ثابت‌های رشته‌ای

تحلیل نحوی: وظیفه‌ی این مرحله بررسی درستی برنامه از نظر نحوی (دستوری) می‌باشد.

* از آن برای تشخیص خطاهای نحوی توسط زبان است. و برای توصیف ساختار نحوی زبان‌های برنامه‌نویسی از برنامه‌های مستقل از متن استفاده می‌کنیم.

خود این مرحله در حد استفاده (تجزیه) می‌باشد که البته وجود خارجی ندارد و به صورت تابعی ساخته می‌شود.

تحلیل معنایی: وظیفه‌ی این مرحله بررسی درستی برنامه از نظر معنایی است.

مثال: int i,j;
string s;
i = j + s; — از نظر لغوی و نحوی درست است — اما معنایی نادرست است

MICRO

int f(int i, float j)

type تابع : int * float → int

Subject : ate

در این مرحله یکسری کنترل های معنایی برای بررسی درستی برنامه از نظر معنایی صورت می‌گیرد. این کنترل های معنایی (semantic checking) به دو دسته تقسیم می‌شوند.

۱) کنترل معنایی ایستا: کنترل هایی هستند که در زمان compile قابل اعمال هستند.

- کنترل نوع ایستا: زبان هایی که دارای strong typing (نوع دهی قوی): قبل استفاده از شناسه در مورد نوع آن باید مشخص شود

- کنترل تعداد و تطابق پارامتر های ظاهری و واقعی توابع

۲) کنترل معنایی پویا: در این روش یکسری کنترل های در زمان اجرا روی برنامه اعمال می‌شود.

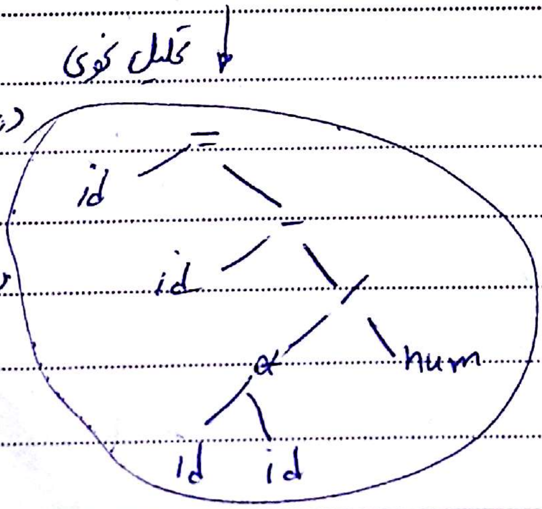
مثال: s = p - q * r / 100 ⇒ از [illegible] چشم‌پوشی می‌کنیم

↓ تحلیل لغوی

id , = , id , - , id , * , id , / , num

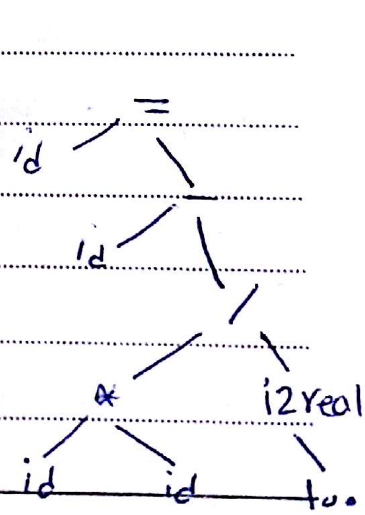
↓ تحلیل نحوی

درخت تولید شکل فشرده از درخت تجزیه



تحلیل معنایی ⇒

با فرض این که s, p, q, r از نوع اعشاری باشند



MICRO

Scanned by CamScanner

شکل کلی دستورات سه آدرسه ی کدمیانی: (opcode, address1, address2, address3)

مستقل از ماشین و ساخت افزار است.

تولید کدمیانی ⟹ (*, q, r, t_1)

(i2real, #100, , t_2)

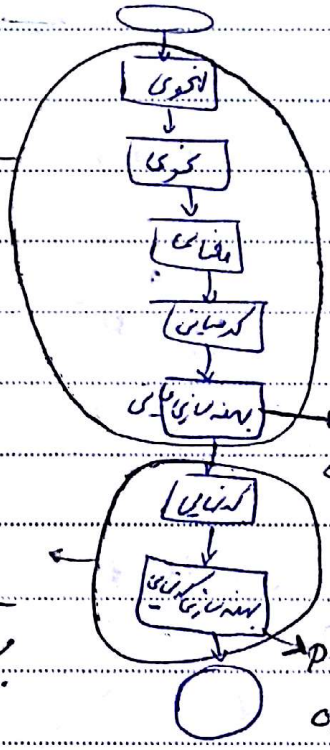
(/, t_1, t_2, t_3)

(-, p, t_3, t_4)

(=, t_4, , s)

t_i ؟ حافظه موقت هستند

جهت نگهداری نتایج میانی ...



مزیت تقسیم:

مثلاً می خواهیم یک زبان را روی چند ماشین مختلف پیاده سازی کنیم، کافی است یک front-end نوشته و چند back-end.

بهینه سازی کدمیانی:

(*, q, r, t_1)

به جای تولید حافظه موقت جدید از t_1 استفاده می کنیم چون t_1 آزاد است.

(/, t_1, #100.0, t_1)

(-, p, t_1, s)

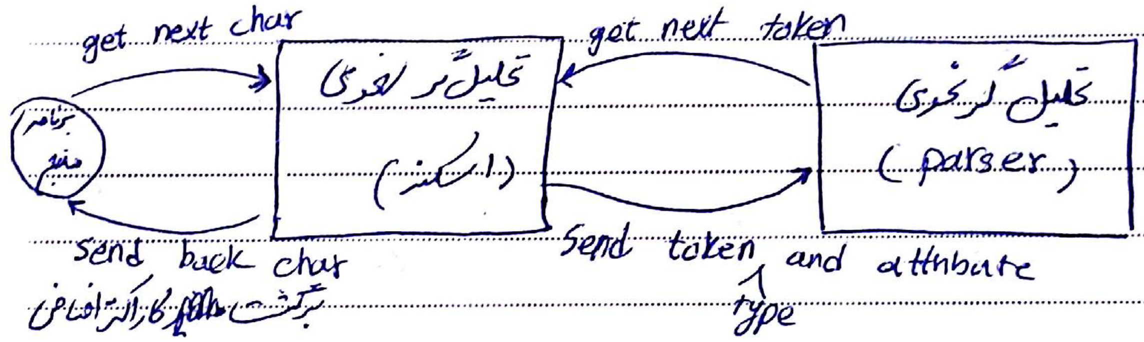
یادآوری می شود که حداقل به یک حافظه موقت نیاز است.

تولید کد نهایی ⟹

```
mov   R1, q
mult  R1, r
div   R1, #100.0
mov   R2, p
sub   R2, R1
mov   s, R2
```

تحلیل لغوی: وظیفه‌ی این مرحله تبدیل برنامه‌ی منبع به دنباله‌ای از نشانه‌های لغوی (توکن) می‌باشد.

شکل زیر نحوه‌ی تعامل مراحل تحلیل لغوی و نحوی را نشان می‌دهد.



مثال: $S = p + q * r$

اولین بار که اسکنر خوانده بشود واژه‌ی از نوع id را شناسایی می‌کند. همیشه در برگشت اسکنر، attribute و token type را برگشت می‌دهد. اگر واژه‌ی شناسایی شده از نوع id باشد، درصورتی که از قبل به آن یک رکورد در جدول نمادها (symbol table) ایجاد نکرده باشد یک رکورد ایجاد کرده و در هر [illegible] نوع توکن id به همراه آدرس رکورد آن در جدول نمادها را به parser برمی‌گرداند.

Subject : Date

جدول نماد: ساختار داده برای نگهداری مشخصات شناسه‌ها

آدرس	واژه	نوع	مقدار اولیه	آدرس حافظه	---
1	S				
2	P				
3	q				
4	r				

برای parser پس از شناسایی واژه‌ی از نوع id، زوج <id, آدرس> ارسال می‌شود.

پس از 8 بار، اسکنر فراخوان شده، دنباله توکن‌های زیر به پارسر ارسال خواهد شد:

<id,1> <=,-> <id,2> <+,-> <id,3> <*,-> <id,4> <;,->

<operator, '='> <operator, '+'> <delimiter, ;>

حالت دوم هم می‌تواند باشد

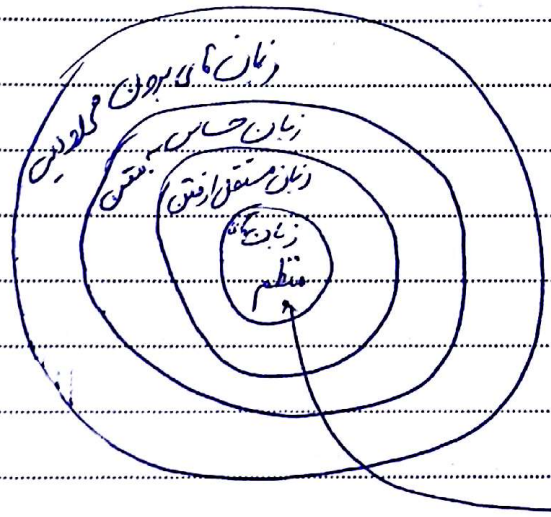
نکته: با توجه به اینکه اسکنر با I/O سروکار دارد از قابلیت زمان‌بر کامپایلرات دو تا توان جهت افزایش سرعت آن از تکنیک بافرینگ استفاده کرد.

روش‌های تولید اسکنر:
- تولید دستی
- تولید خودکار با استفاده از ابزار LEX، ALEX، JLEX، ANTLEX

Subject : ate

تولید رشته‌ها به‌طوری که هریک از انواع توکن‌ها (رشته‌ها برای ...) یک زبان منظم باشند

روش‌های توصیف و تشخیص زبان‌های منظم می‌توان برای توصیف و تشخیص توکن‌ها استفاده کرد



برای ساخت اسکنر

- روشی برای توصیف زبان‌های منظم: عبارت منظم
- تشخیص: DFA / NFA → FA
- تولید: برنامه منظم

عبارت‌های منظم: $\Sigma = \{a, b\}$

مثال: یک عبارت منظمی بنویسید که رشته‌های متشکل از a و b که شامل bbb باشد؟ مثال 1)

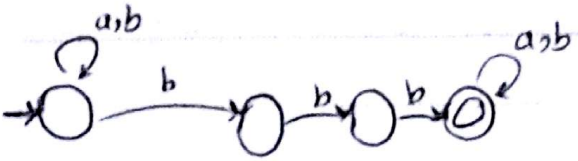
$$(a+b)^* bbb (a+b)^*$$

مثال 2) که شامل bbb نباشد

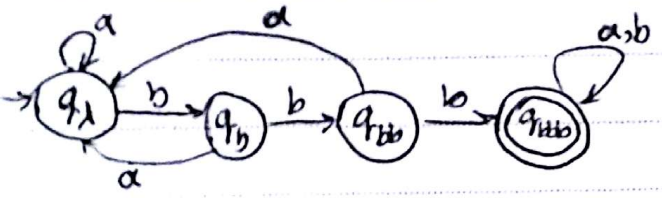
$$(a+ba+bba)^* (\lambda+b+bb)$$

Subject : Date

ماشین ← قطعی / غیرقطعی ← مثال:



برای NFA !



برای DFA !

توصیف توکن ها با استفاده از عبارت های منظم

letter a|b|c|...|z|A|B|...|Z

digit 0|1|2|3|...|9

id letter (letter | digit)*

num (+|-|λ) digit digit* (. digit digit*)(E (+|-|λ) digit digit* | λ)

قواعد خلاصه سازی $r? \equiv r|\lambda$, $r^+ \equiv rr^*$, $[abc] \equiv a|b|c$

پیاده سازی این خلاصه سازی ها ⟸

letter [a-zA-Z]

digit [0-9]

id letter (letter | digit)*

num (+|-)? digit+ (. digit+)? (E (+|-)? digit+)?

Subject : ate

مبانی از تولید سی اسکیز (تحلیل گرلغوی)

block → { stmts }

stmts → stmt stmts | stmt | block

stmt → asign_stmt | for_stmt | while_stmt | if stmt

asign_stmt → id = expr ;

for_stmt → for (expr ; expr ; expr) stmt

if_stmt → if expr stmt | if expr stmt else stmt

while_stmt → while expr stmt ...

- از روی برنامه زبان ابتدا لیست تمامی پایانه ها را استخراج می کنیم. پایانه همان توکن هاهستند
- تمامی توکن ها را دسته بندی کردی تا انواع تمامی توکن ها مشخص شود.
- الگوی تمامی توکن ها را با استفاده از عبارت منظم توصیف می کنیم.
- سپس عبارت های منظم را به ماشین متناهی غیرقطعی تبدیل می کنیم.
- ماشین متناهی را قطعی (ولمینه) تبدیل می کنیم.
- پس ماشین متناهی قطعی حاصل را پیکر تبدیل می کنیم تا برنامه اسکنر ساخته آید.

MICRO

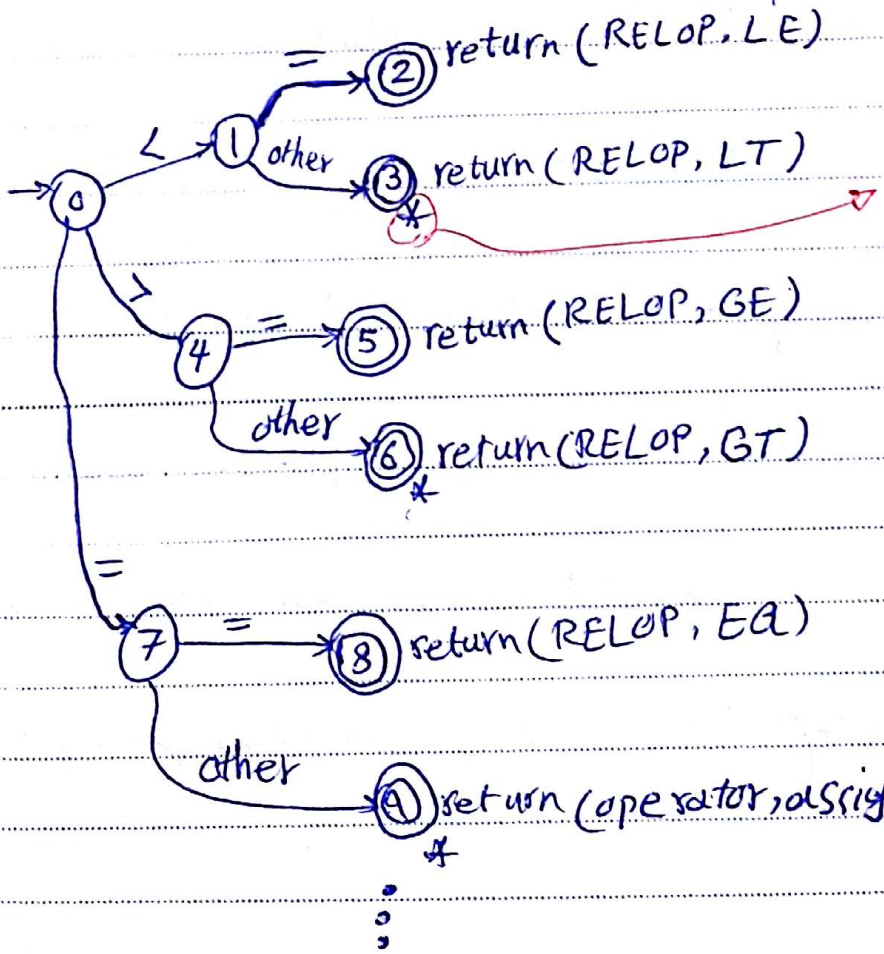
Subject : Date

Terminals = { '{', '}', id, =, ;, for, '(', ')', if, else, ---}

انواع توکن ها:

نمونه	الگو	نوع توکن
if, ...	if|else|for|while	کلمات کلیدی
tax, il, ...	$\ell(\ell\|d)^*$	id
	(|)|{|}|;	جداکننده
10e, 17.5, --	$d^+(.d^+)?(E(+\|-)?d^+)$	num
	<, <=, >, >=, ==, !=	عملگر
⋮	⋮ relational operator	⋮

Less or equal



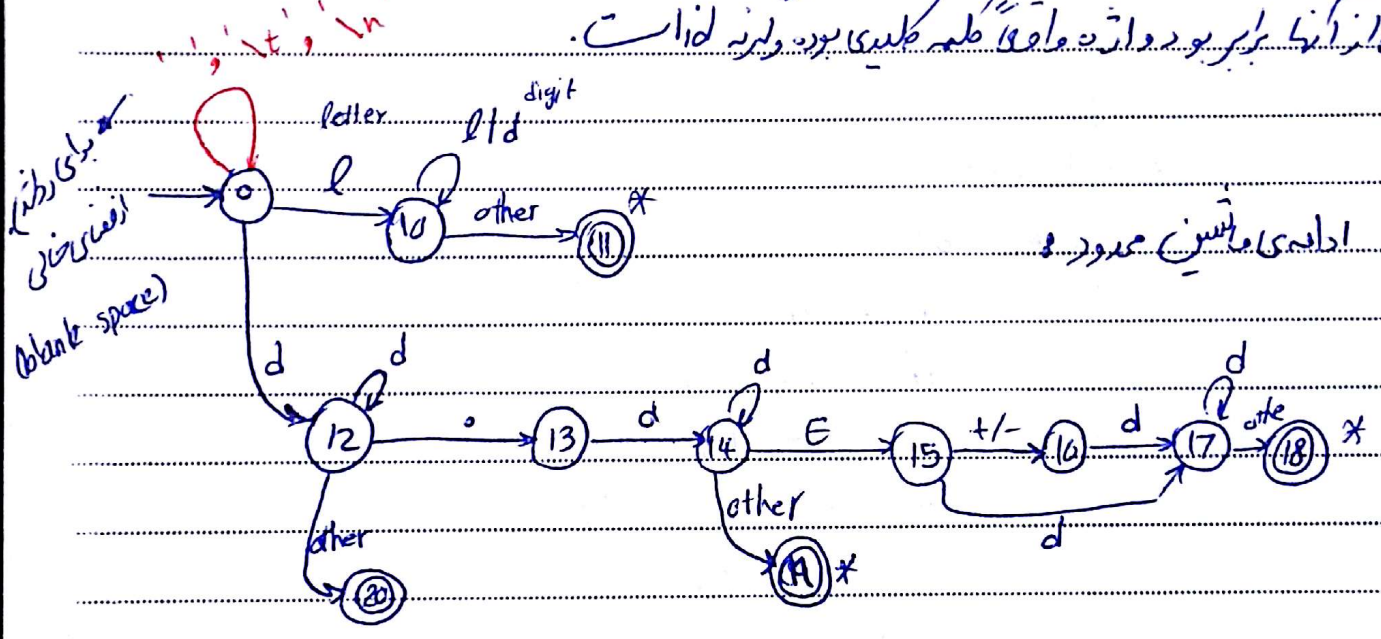
قرارداد: آخر کاراکتر اضافی خواندیم * می‌گذاریم

Subject : Date

- برای تشخیص کلمات کلیدی دو راه وجود دارد:

1) استفاده از همان ماشین محدود شناسه که وقتی یک واژه با الگوی id پیدا شد واژه را با لیستی از کلمات کلیدی که در یک داده ساختار (مثلاً آرایه یا رشته‌ها) ذخیره کردیم مقایسه می‌کنیم تا ببینیم آیا یکی از آنها برابر بود واژه ما واقعاً کلمه کلیدی بوده و گرنه id است.

ادامه‌ی ماشین محدود:



2) برای کلمات کلیدی نیز ماشین محدود طراحی کرده و در بالای ماشین id قرار می‌دهیم. ماشین حاصل ممکن است غیرقطعی باشد که می‌توان آن را به قطعی معادل تبدیل کرد.

* برنامه‌ی اسکنر چیزی نیست جز برنامه‌ی پیاده‌سازی ماشین متناهی قطعی حاصل

- دو روش برای پیاده‌سازی DFA حاصل وجود دارد:

Subject :

Date :

1) پیاده سازی با استفاده از ساختار شرطی مانند switch_case

2) جدول انتقال " " "

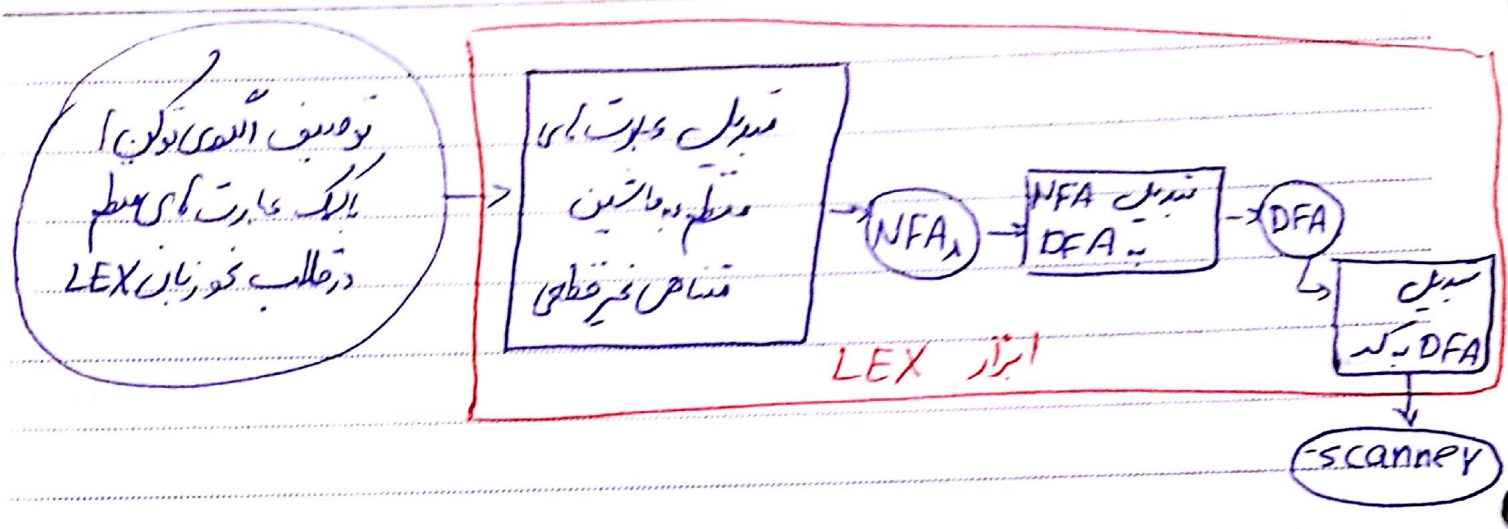
1) int scanner(){

```
int state;
char ch;
while (1){
    switch (state){
        case 0:
            nextchar(ch);
            if (ch == '<')
                state=1;
            else if (ch == '>')
                state=4;
                :
            break;
        case 1;
                :
    }
}
```

	<	>	=	
0	1	4	7	
1			2F	
2	:			

جدول انتقال

State = DFA_TABLE[state, ch]



تمرین فصل 2: 1، 2 (الف، ج، د)، 3 (الف، ب، ج، د)، 7، (ز، ح)، 8، 17، 15

جلسه 12/8/95

* تا رشته ای که تعداد ab و ba برابرند را عبارت منظم بنویسید؟ (با حرف یکسان شروع و خاتمه می یابد)

$$a(a+b)^*a + b(a+b)^*b + \lambda + a + b$$

* ابزار ANTLR، به صورت اتوماتیک تحلیلگر لغوی و نحوی را تولید می کند.

تحلیل نحوی:

برای توصیف ساختار نحوی زبان های برنامه نویسی از گرامرهای مستقل از متن استفاده می کنند.

شکل قواعد: $A \to \alpha \quad , A \in V \quad , \alpha \in (V \cup T)^*$

استخراج: چپ (LMD) — Left most derivation

راست (RMD) — Right most derivation

درخت اشتقاق: مثال ←

$G: S \to AB$

$A \to aAb \mid \lambda$

$B \to cB \mid g$

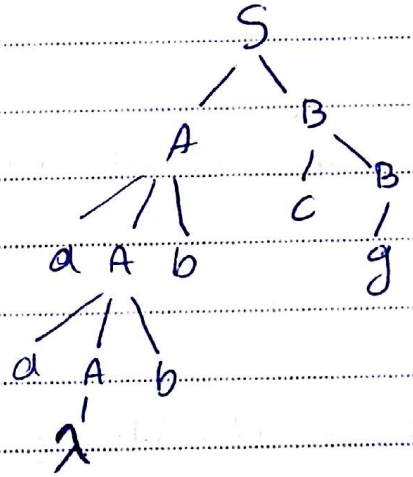
$w = aabbcg$

چپ‌ترین را اول جایگذاری می‌کنیم.

$LMD: S \Rightarrow AB \Rightarrow aAbB \Rightarrow aaAbbB \Rightarrow aabbB \Rightarrow aabbcB \Rightarrow aabbcg$ ✓

$S \Rightarrow AB \Rightarrow a^n A b^n B \Rightarrow \dots \Rightarrow a^n b^n c^m g$ ← چه زبانی تولید می‌کند؟

$L(G) = \{ a^n b^n c^m g \mid n, m \geq 0 \}$

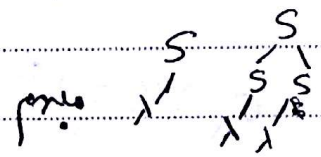


درخت اشتقاق ←

* درخت اشتقاق برای اشتقاق چپ و راست یکی است

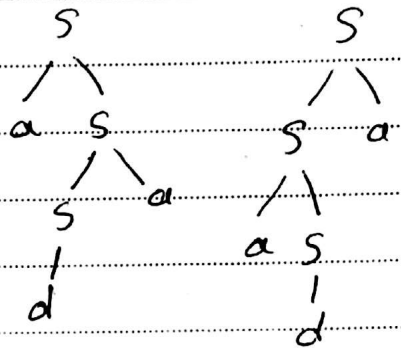
قرار می‌دهیم: گرامری مبهم است اگر رشته‌ی $w \in L(G)$ وجود داشته باشد به طوری که برای آن دو LMD متفاوت (یا دو RMD متفاوت یا دو درخت اشتقاق متفاوت) داشته باشیم.

مثال: $G_1: S \to aSb \mid SS \mid \lambda$ مبهم



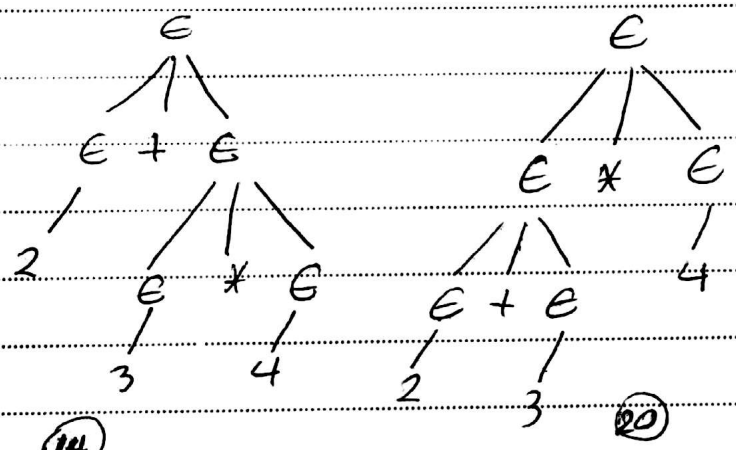
$G_2: S \to aSb \mid \lambda$ غیرمبهم

$G_3 : S \rightarrow aS \mid Sa \mid d$ $\quad w = ada$



$G_4 : E \rightarrow E+E \mid E*E \mid 2|3|4|5$

$w = 2+3*4$



(14) (20)

statement

$G_5 : St \rightarrow$ if expr then St

| if expr then St else St

| a

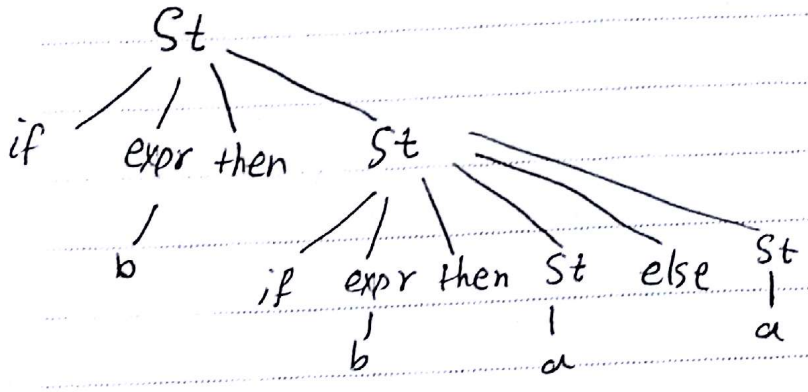
expr $\rightarrow$ b

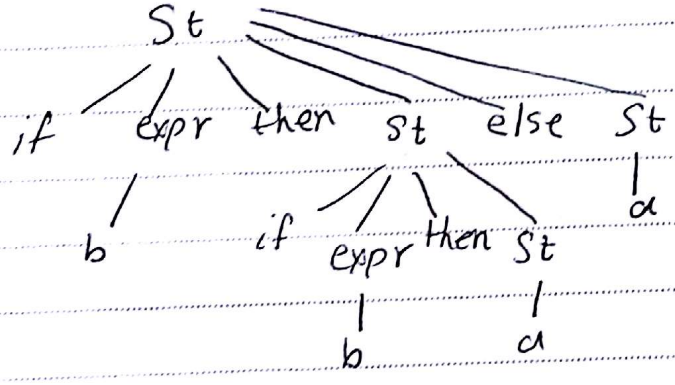
w = if b then if b then a else a

- دو درخت در صفحه بعد هستند هردو w را می سازند پس G_5 مبهم است.

Subject :

Date





$G_8: E \to E+E \mid E \neq E \mid E/E \mid E-E \mid -E \mid E \hat{} E \mid (E) \mid id \mid num$

شرکت پذیری چپ - و + دودویی

" " " / و ≠ "

" راست ^ "

" " " یکانی - "

" " " ( ) "

G_8 مبهم است. با freeze کردن

یک گرامر جدید معادل با G_8 ی بازنویسی

که مبهم نباشد.

$\hat{G}_8 : E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to P \wedge F \mid P$

$P \to -P \mid M$

$M \to (E) \mid num \mid id$

$G_8 \equiv \hat{G}_8$

G_8 ↓ مبهم $\hat{G}_8$ ↓ غیرمبهم

نکته: الگوریتمی برای تعیین مبهم بودن یا نبودن گرامر مستقل از متن وجود ندارد.

مسئله ابهام در حوزه‌ی زبان‌ها و گرامرهای مستقل از متن مطرح است.

گرامر مستقل از متن ← مبهم / غیرمبهم

زبان مستقل از متن ←
- ذاتاً مبهم: هر گرامری که برای آن می‌نویسیم مبهم است.
- غیرمبهم: اگر بتوان یک گرامر مستقل از متن غیرمبهم برای آن نوشت.

$L = \{a^n b^n c^m : n,m \geq 0\} \cup \{a^n b^m c^m : n,m \geq 0\}$

↓ ذاتاً مبهم

تمرین فصل 3: ص 143، 1 و 2 و 3 ج و هـ و و ز

Subject :

Date 1395, 8, 19

$first(\alpha) = \{a \in T : \alpha \overset{*}{\Rightarrow} a\beta, \alpha, \beta \in (V \cup T)^*\}$

$follow(A) = \{b \in T : S \overset{*}{\Rightarrow} \alpha A b \beta, \alpha, \beta \in (V \cup T)^*\}$

الگوریتم

مثال:

G: $S \to aA \mid bB$

LL(1) $A \to aA \mid b$

$B \to bB \mid d$

قواعد S: $S \to aA \mid bB$ $\quad S \Rightarrow aA$, $S \Rightarrow bB$

$first(S) = \{a\} \cup \{b\} = \{a, b\}$

$first(aA) = \{a\}$ $\quad first(aA) \cap first(bB) = \phi$

$first(bB) = \{b\}$

قواعد A: $A \to aA \mid b$

$first(A) = first(aA) \cup first(b) = \{a\} \cup \{b\} = \{a, b\}$

$first(aA) \cap first(b) = \phi$

قواعد B: $B \to bB \mid d$ $\quad first(B) = \{b, d\}$

$first(bB) \cap first(d) = \phi$

Subject:

Year: Month: Day: ( )

پیشگو ← LL(1)

$S \to AB \mid CD$ — $first(A) = \{a, b\}$ — گرامر $G(*)$

$A \to aA \mid b$ — $first(B) = \{b, e\}$

$B \to bBb \mid e$ — $first(C) = \{d, g\}$

$C \to dC \mid g$ — $first(D) = \{g, e\}$

$D \to gD \mid e$ — $first(S) = \{a, b, d, g\}$

$first(AB) \cap first(CD) = \phi$

* پس اگر در گرامر قواعدی به شکل $A \to \alpha \mid \beta$ داشته باشیم، باید اشتراک $first(\alpha)$ و $first(\beta)$ تهی باشد.

پارسر بازگشتی پیشگو:

در این پارسر برای هر متغیر یک تابع می‌نویسیم که معمولاً بازگشتی است. در بدنه‌ی هر تابع ابتدا با استفاده از توکن جاری قاعده‌ی صحیح صحت تجزیه‌ی آن متغیر را انتخاب کرده و سپس طبق آن مدل تجزیه را دنبال می‌کنیم. همچنین یک تابع اضافی به نام match داریم که بدنه‌ی آن به صورت زیر است.

```
match (t : token) {
    if (lookAheadtoken = t) then
        lookAheadtoken = nexttoken();
    else
        SyntaxError();
}
```

Subject:

Year: Month: Day: ( )

برای تابع S(x)

```
S(){
 if lookAheadToken is in {a,b} then {
        call A(); call B();
 }else if lookAheadToken is in {d,g} then{
        call C(); call D();
 }else
        SyntaxError();
}

A() {        ↗ lookAheadToken
    if LA is in {a} then{
        match('a'); call A();
 }else if LA is in {b} then
        match('b');
  else  SyntaxError();
```

و همین شکل برای B و D, C

مشکل بازگشتی چپ در پارسرهای بالا به پایین:

یادآوری: بازگشتی چپ $A \to A\alpha$

و راست $A \to \alpha A$

Subject:

Year: Month: Day: ( )

مثال: $G: A \to Ab \mid d$

$L(G) = L(db^*)$

```
A() {
  if LA is in {d} then {
    call A(); match('b');
  }
  else if ---
};
```

ایجاد حلقه برای

بازگشتی چپ باید حذف شود.

حذف بازگشتی چپ مستقیم:

$G: A \to Ab \mid d$

$\hat{G}: A \to dA'$

$A' \to bA' \mid \lambda$

* شکل کلی حذف بازگشتی چپ مستقیم:

$G: A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$

$\hat{G}: A \to \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \lambda$

Subject:

Year: Month: Day: ( )

مثال: تبدیل بازگشتی چپ غیر مستقیم به بازگشتی راست

$G: S \to Ab \mid e$

$A \to Sa \mid d$ ⟹ (جایگزینی) $\hat{G}: S \to Ab \mid e$

$A \to Aba \mid ea \mid d$

(α_1: aba, β_1: ea, β_2: d)

$\hat{\hat{G}}: S \to Ab \to e$

$A \to eaA' \mid dA'$

$A' \to baA' \mid \lambda$

$G \equiv \hat{G} \equiv \hat{\hat{G}}$

نکته: شرط لازم برای LL(1) بودن (و به طور کلی LL(k) بودن) یک گرامر، حذف بازگشتی چپ (در صورت وجود) است.

تمرین: 4

جلسه 95/8/26

فاکتورگیری چپ:

مثال: $G: A \to aaab \mid aaad \mid g$

$G \equiv \hat{G}$

$\hat{G}: A \to aaaA' \mid g$

$A' \to b \mid d$

Subject:

Year: Month: Day: ( )

شکل کلی اعمال فاکتوریزی چپ:

$G: A \to \alpha\beta_1 | \alpha\beta_2 | \cdots | \alpha\beta_n | \gamma_1 | \gamma_2 | \cdots | \gamma_m$

اعمال فاکتوریزی چپ ↓

$\hat{G}: A \to \alpha A' | \gamma_1 | \gamma_2 | \cdots | \gamma_m$

$A' \to \beta_1 | \beta_2 | \cdots | \beta_n$

الگوریتم کوی follow ، first

الگوریتم first:

تعریف تابع first

$first(\alpha) = \{a \in T : \alpha \overset{*}{\Rightarrow} a\beta , \alpha, \beta \in (V \cup T)^* \}$

و اگر $\alpha \overset{*}{\Rightarrow} \lambda$ ، λ را نیز به $first(\alpha)$ اضافه می‌کنیم.

مثال:

$S \to ABCD$ $\quad first(S) = \{a, b, d, g, \lambda\}$

$A \to aA | \lambda$ $\quad first(A) = \{a, \lambda\}$

$B \to bB | \lambda$ $\quad first(B) = \{b, \lambda\}$

$C \to dC | \lambda$ $\quad first(C) = \{d, \lambda\}$

$D \to gD | \lambda$ $\quad first(D) = \{g, \lambda\}$

از پایین به بالا

TANDIS

Subject: .... Year: ...... Month: ...... Day: ...... ( )

مثال:

$S \to ABCeD \mid g$ — $first(S) = \{g, a, b, m, d, e\}$

$A \to Aa \mid \lambda$ — " $(A) = \{a, \lambda\}$

$B \to bB \mid \lambda$ — " $(B) = \{b, \lambda\}$

$C \to Cd \mid mC \mid \lambda$ — " $(C) = \{\lambda, m, d\}$

$D \to Dn \mid gD \mid \lambda$ — " $(D) = \{g, n, \lambda\}$

مثال:

$S \to ABC \mid Sg$ — $first(S) = \{a, e, b, n, d, \lambda, g\}$

$A \to AMe \mid \lambda$ — " $(A) = \{\lambda, a, e\}$

$M \to Ma \mid \lambda$ — " $(M) = \{a, \lambda\}$

$B \to bB \mid D$ — " $(B) = \{b, n, \lambda\}$

$C \to Cd \mid \lambda$ — " $(C) = \{d, \lambda\}$

$D \to Dn \mid \lambda$ — " $(D) = \{n, \lambda\}$

الگوریتم follow

تعریف follow:

$$follow(A) = \{ b \in T : S \overset{*}{\Rightarrow} \alpha A b \beta,\ \alpha, \beta \in (V \cup T)^*,\ A \in V \}$$

نماد شروع

$$B \to \alpha A \beta \quad (\alpha, \beta \in (V \cup T)^*)$$

تمام عناصر first(B) را به follow(A) اضافه می کنیم

TANDIS

مثال:

$S \longrightarrow ABC$ follow(S) = { $ }

$A \rightarrow Aa \mid bA \mid \lambda$ " (A) = {a, d, g, e, $}

$B \rightarrow Bd \mid \lambda$ " (B) = {g, e, d, $}

$C \rightarrow gC \mid Ce \mid \lambda$ " (C) = {e, $}

مثال:

$S \rightarrow ABC \mid Sg$ follow(S) = {$, g}

$A \rightarrow aA \mid \lambda$ " (A) = {n, a, b, d, m, e, g, $}

$B \rightarrow Bb \mid AM$ " (B) = {b, d, m, e, g, $}

$M \rightarrow n \mid \lambda$ " (M) = {g, $, d, m, e, b}

$C \rightarrow CDe \mid \lambda$ " (C) = {d, m, e, g, $}

$D \rightarrow dD \mid Dm \mid \lambda$ " (D) = {m, e}

نکته: $B \rightarrow AM$ آمد $M \rightarrow \lambda$ داشتیم هرچه بعد از B می‌باشد (follow(B)) باید follow(A) اضافه شود.

شرایط LL(1) بودن یک گرامر:

کافی برای

- برای هر جفت قاعده‌ی گرامر به شکل $A \to \alpha | \beta$ باشد، باید دو شرط زیر برقرار باشد:

1) اشتراک first(α) و first(β) تهی باشد

2) اگر $\beta \Rightarrow^* \lambda$، آنگاه اشتراک follow(A) و first(α) باید تهی باشد.

مثال:

$G_1: S \to aAb$ ← LL(1) است.
$A \to bA | \lambda$ شرط ② را ندارد.

$G_2: S \to aAd$
$A \to bA | \lambda$ ✓ ← LL(1) است.

$G_3: S \to AB | dC$ ← LL(1) نیست.
$A \to aA | \lambda$ AB می‌تواند λ شود بنابراین طبق شرط 2 باید با first(dC) ∪ follow(S) برابر با ϕ باشد. (که این‌گونه نیست.)
$B \to bB | \lambda$
$C \to ge | esd | \lambda$

Subject:

Year: Month: Day: ( )

* اگر گرامری $LL(1)$ نباشد و $LL(1)$ معادل برای آن وجود داشته باشد با استفاده از تکنیک های زیر می توان آن را به $LL(1)$ تبدیل کرد:

1) حذف بازگشتی چپ در صورت وجود

2) فاکتورگیری چپ در صورت نیاز

3) رفع ابهام در صورت مبهم بودن

4) استخراج زبان برای نوشتن یک گرامر $LL(1)$ معادل برای آن

مثال: $G: S \to aAd$

$A \to dA \mid \lambda$

$L(G) = L(ad^*d) = L(add^*)$

$\hat{G}: S \to adB$

$B \to dB \mid \lambda$

تمرین 4، 6، 7، 10، 11

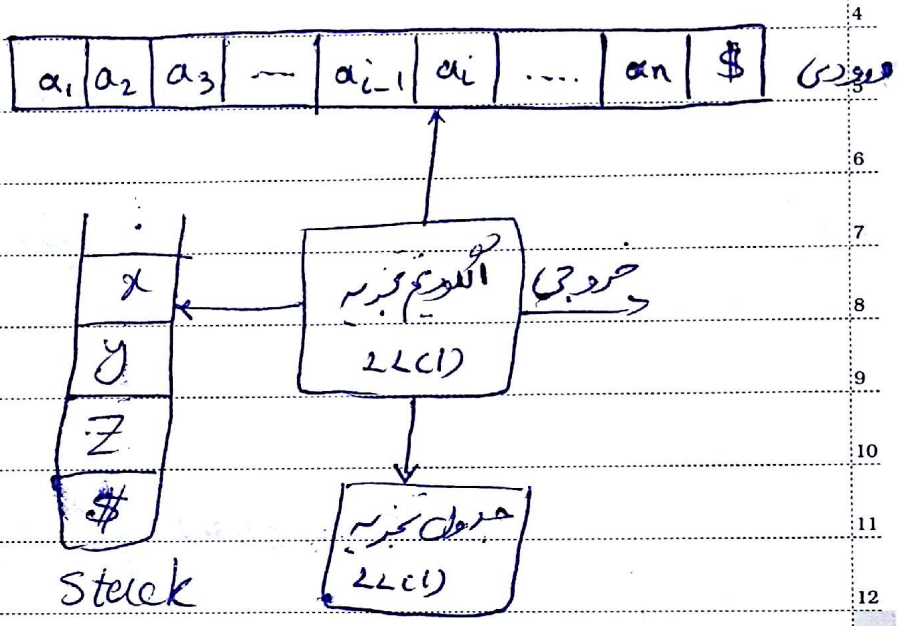
ج 7، جلسه درلوشی موجود است

Subject:

Year: Month: Day: ( )

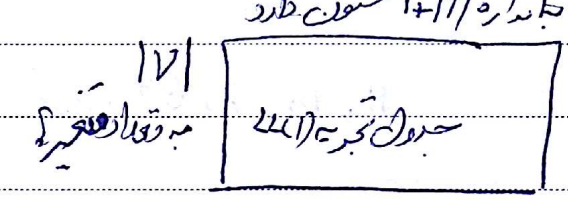
پارسر غیربازگشتی پیشگو LL(1)

ساختار پارسر LL(1) به صورت زیر است.



1/2

در جدول تجزیه LL(1) برای انتهای هر قاعده‌ی صحیح استفاده می‌شود. این جدول به تعداد ترمینال‌ها + یک پایانه (متغیر؟) سطر داشته و به تعداد پایانه + 1 (ستون اضافی برای $ می‌باشد) ستون دارد.



$|PT| = |V| \times (|T|+1)$

ساخت جدول تجزیه LL(1)

الگوریتم‌های مرتبط را شماره‌گذاری می‌کنیم.

Subject: .....

Year:..... Month:..... Day:.....( )

برای هر قاعده به شکل $A \to \alpha$ دو قدم زیر را اعمال می‌کنیم

1) در سطر غیرپایانه A و زیر ستون‌های عناصر $first(\alpha) - \{\lambda\}$ شماره قاعده $A \to \alpha$ را قرار می‌دهیم

2) اگر $\alpha \overset{*}{\Rightarrow} \lambda$ (α مستقیم یا غیرمستقیم λ را تولید کند) در سطر A زیر ستون‌های عناصر $follow(A)$ شماره قاعده $A \to \alpha$ را قرار می‌دهیم.

مثال: 1) $S \to aAb$

2,3) $A \to bA \mid \lambda$

	a	b	$
S	①		
A		②,③	

یک خانه 2 قاعده وجود دارد

پس ترا (1) LL نیست.

مثال: 1) $S \to aAd$

2,3) $A \to bA \mid \lambda$

$follow(A) = \{d\}$

	a	b	d	$
S	①			
A		②	③	

Subject:

Year: Month: Day: ( )

Jus: G : S → AB ① first(S) = {a, b, c, λ} follow(S) = {$}

2, 3) A → aA | bC " (A) = {a, λ, c} " (A) = {b, $}

4, 5 B → bB | λ " (B) = {b, λ} " (B) = {$}

6, 7 C → cC | λ " (C) = {c, λ} " (C) = {b, $}

	a	b	c	$
S	1	1	1	1
A				
B				
C				

Jus: ① E → TE' follow(E) = {$,)}

2, 3) E' → +TE' | λ " (E') = {$,)}

4) T → FT' " (T) = {+, $,)}

5, 6 T' → *FT' | λ " (T') = {+, $,)}

7, 8 F → (E) | id " (F) = {*, +, $,)}

	+	*	(	)	id	$
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Subject: .....

Year: ..... Month: ..... Day: ..... ( )

الگوریتم تجزیه LL(1) (الگوریتم در کتاب هست، بایک مثال الان بررسی می‌کنیم)

$E\$ \xrightarrow{①} TE'\$ \xrightarrow{④} FTE'\$ \xrightarrow{⑧} \dots \rightarrow id+id*id\$$

بخش تجزیه شده	پشته	بخش تجزیه نشده	عمل
—	$E\$$	$id+id*id\$$	① خروجی
—	$TE'\$$	$id+id*id\$$	④ خروجی
—	$FTE'\$$	$id+id*id\$$	⑧ خروجی
—	$idT'E'\$$	$id+id*id\$$	match
id	$T'E'\$$	$+id*id\$$	⑥ خروجی، $T \rightarrow \lambda$
id	$E'\$$	$+id*id\$$	
⋮	⋮	⋮	⋮
$id+id*id$	$\$$	$\$$	accept

w$

تمرین فصل 3: 5, 8, 12, 13, 14, 16 (الف و ب)

Subject:

Year: Month: Day: ( )

تجزیه پایین به بالا

در این روش کلی تجزیه، درخت تجزیه از برگ‌ها به ریشه ساخته می‌شود و ترتیب به کارگیری قواعد معمولاً عکس RMD است.

مثال:

$G: E \to E+T \mid T$ 1,2

$T \to T*F \mid F$ 3,4

$F \to (E) \mid id \mid num$ 5,6,7

$w = id+id*id$

$RMD: E \overset{1}{\Rightarrow} E+T \overset{3}{\Rightarrow} E+T*F \overset{6}{\Rightarrow} E+T*id \overset{4}{\Rightarrow} E+F*id \ldots \overset{6}{\Rightarrow} id+id*id$

بازسر پایین به بالا

E

$\underline{E+T}$

$E+\underline{T*F}$

$E+\underline{T}*id$

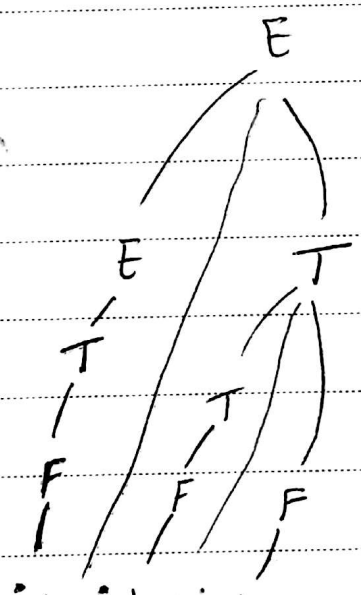
$E+\underline{F}*id$

$\underline{E}+id*id$

$\underline{T}+id*id$

$\underline{F}+id*id$

$id+id*id$



درخت پایین به بالا

TANDIS

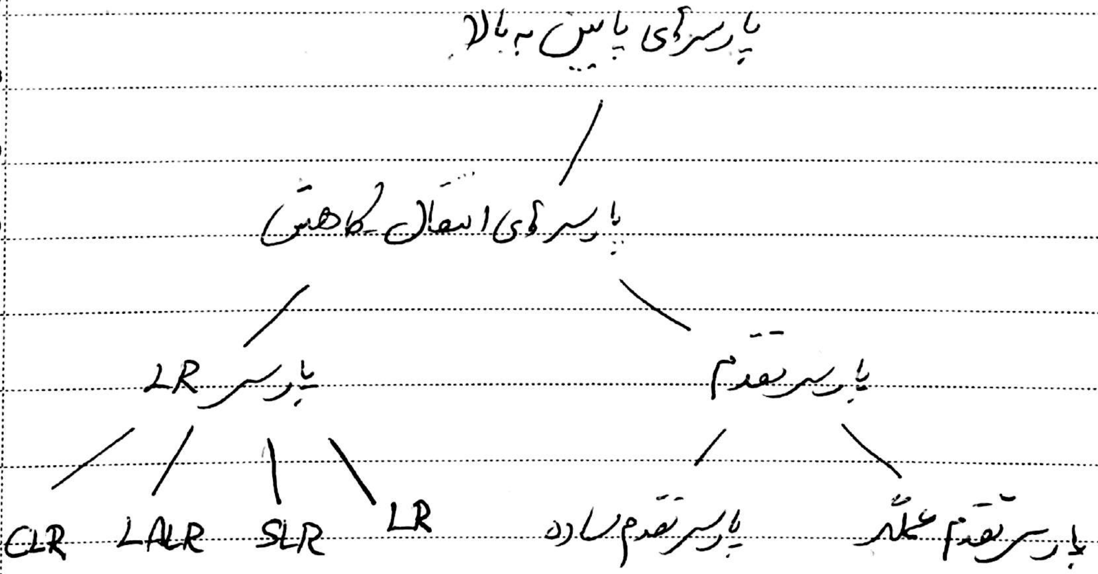
Subject:
Year: Month: Day: ( )

به هر یک از عبارت هایی که خط زیر آن کشیده شده و دقیقاً با سمت راست یکی از قواعد برابر است هندل یا دستگیره می گویند.

در واقع در روش های تجزیه پایین به بالا دنبال یافتن دستگیره هستیم. در این روش های تجزیه آن قدر عمل shift (یعنی انتقال توکن جاری به بالای پشته) صورت می گیرد تا در بالای پشته یک دستگیره یافت شده و طبق آن عمل کاهش (reduce) صورت گیرد و آن قدر عمل shift و reduce صورت می گیرد تا در صورت امکان به ریشه برسیم

به این پارسرها پارسرهای انتقال - کاهش (shift - reduce parser) گویند

دسته بندی پارسرهای انتقال - کاهش:



Subject:
Year: Month: Day: ( )

به طور کلی در این پارسر ۴ عمل (action) انجام می شود

- عمل انتقال (shift): توکن ورودی را در بالای پشته قرار می دهد و هم چنین اسکنر جهت دریافت توکن بعدی فراخوانی می شود.

- عمل کاهش (reduce): یک دستگیره در بالای پشته یافت شود و طبق آن باید عمل کاهش صورت گیرد (یعنی دستگیره از بالای پشته pop شده و غیرپایانه سمت چپ قاعده‌ای که سمت راست آن با دستگیره برابر است push شود.

- عمل accept: خاتمه‌ی تجزیه با موفقیت

- عمل کشف خطا: کشف خطای نحوی و فراخوانی روتین خطا

مثال: جدول برای مثال قبلی

Stack	بخش تجزیه نشده	عمل
-	↗id+id*id$	shift
id	↗+id*id$	reduce by (6)
F	↗+id*id$	reduce by (4)
T	↗+id*id$	reduce by (3)
E	↗+id*id$	shift
E+	↗id*id$	shift
⋮	⋮	⋮
E+T	↗$	reduce by (1)
E	↗$	accept

Subject:

Year: Month: Day: ( )

* در این پارسر ۲ نوع تداخل می تواند رخ دهد:

1) تداخل انتقال-کاهش S/r (Shift/reduce)

2) تداخل کاهش-کاهش r/r (reduce/reduce)

تداخل S/r: در این حالت پارسر به جایی برسد که هم می تواند عمل shift انجام دهد هم عمل reduce و پارسر نمی داند کدام یک را انجام دهد.

مثال: گرامر مفهوم ساختار شرطی را به صورت زیر بیان:

St → if expr then St

| if expr then St else St

| a

Expr → b

ورودی: if b then if b then a else a

stack	ورودی	action
-	if b then if b then a else a $	shift
⋮	⋮	
if expr then if expr then St	else a $	shift / reduce

تداخل ۲/۲:

این تداخل به دو شکل می‌تواند رخ دهد:

parser(a یک دستگیره در بالای پشته پیدا کرده که با سمت راست بیش از یک قاعده

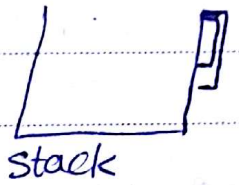
تراورات و نمی‌داند به کجای پایانه سمت چپ کدام یک از قواعد آن را کاهش دهد.

$T \to id$

$F \to id$

parser(b هر دستگیره متناظر در بالای پشته پیدا کرده و می‌داند طبق کدام از آن‌ها

حمل کاهش را انجام دهد.



پارسرهای LR

RMD in reverse

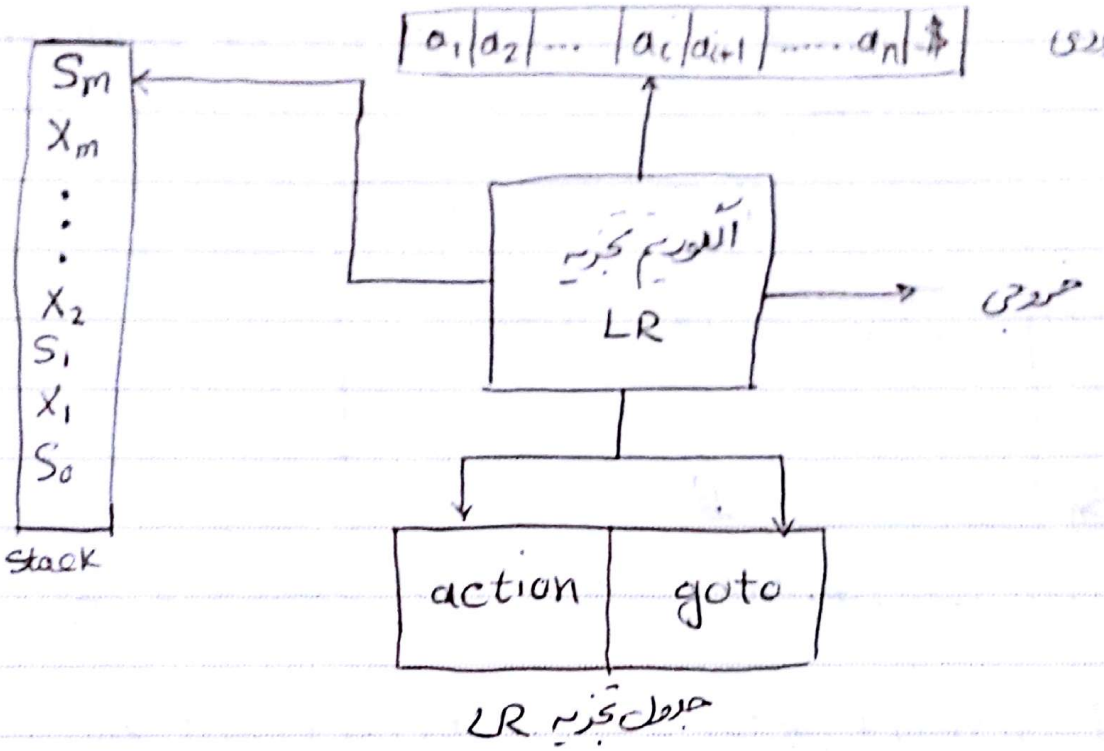
LR(k) است

parser، حالت خاصی از LR(1) ، parser

Left to right

k مقدار توکن جهت انتخاب عمل صحیح می‌باشد

ساختار پارسر LR(1) بصورت زیر است.



از پشته برای نگهداری فرم جمله‌ای راست منطقی استفاده می‌شود. محتوای پشته بصورت زیر است:

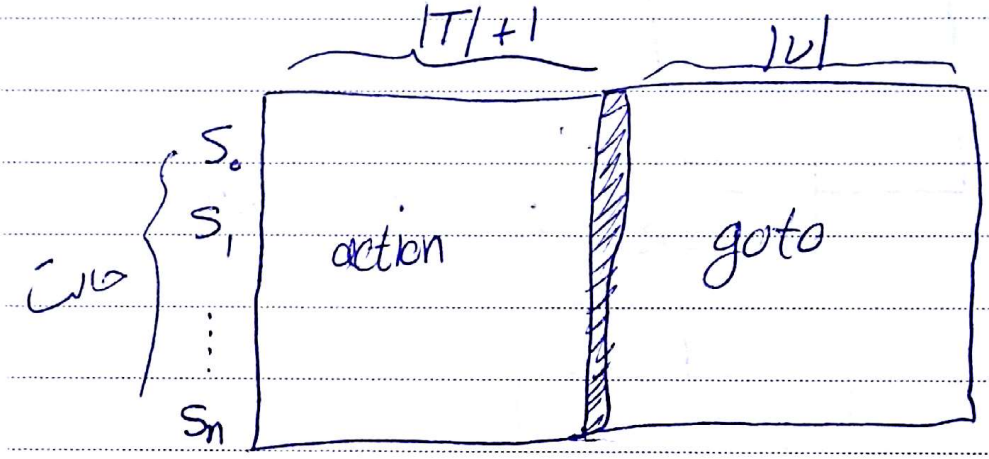
$$S_0 X_1 S_1 X_2 S_2 X_3 S_3 \dots X_m S_m$$

درحالی که S_i ها بیان‌گر حالت (state) یا وضعیت هستند و X_j ها بیانگر علائم ترمیالی (پایانه یا ثابت) می‌باشند.

جدول تجزیه از دو بخش action و goto استفاده شده است.

از بخش action برای انتخاب حل صحیح استفاده می‌شود و به تعداد حالات سطر داشته و به

تعداد پایانه‌ها $S + 1$ ستون دارد (ستون اضافی برای # می‌باشد)



بخش goto به تعداد حالات سطر داشته و به تعداد متغیرها ستون دارد.

$|PT| = |action| + |goto| =$ تعداد حالات $*(|T|+|V|+1)$

هزینه action بصورت زیر است:

هر خانه‌ی goto یا خانه‌ای که حاوی یک شماره حالت است یا خالی است

پایانه

حالت

shift → S_i → حالت

or

reduce → r_j → شماره قاعده

or

acc

or

پایانه‌ها خطای نحوی → خالی است

Subject:

Year: Month: Day: ( )

مثال:

$E \rightarrow E$

$E \rightarrow E+T \mid T$ 1,2

$T \rightarrow T*F \mid F$ 3,4

$F \rightarrow (E) \mid id$ 5,6

جدول تجزیه SLR(1) برای قواعد (ص 198,7 کتاب)

State	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				acc			
2		r2	S7		r2	r2			
3		r4	r4		r4	r4			
4	S5			S4			8	2	3
5		r6	r6		r6	r6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		r1	S7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

از gato بدستی آید

Stack	ورودی	کل
0	id + id * id $	shift (S5)
0 id 5	+ id * id $	reduce by (6)
0 F (3)	+ id * id $	reduce by (4)
0 T 2	+ id * id $	reduce by (2)
0 E 2	+ id * id $	shift (S6)
0 E 1 + 6	id * id $	shift (S5)
0 E 1 + 6 id 5	* id $	reduce by (6)
⋮	⋮	⋮

ساخت جدول تجزیه SLR(1) (در کتاب موجود است) (عکس در گوشی)

فصل 4

تمرین: 2، 13، 22

مثال: ساخت جدول تجزیه SLR(1)

در کتاب (صـ 206) مثال 4-17

Subject:

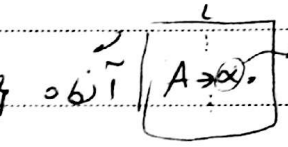
Year: Month: Day: ( )

ساخت جدول تجزیه LR(0)

ابتدا باید نمودار انتقال LR(0) ساخته شود که همان نمودار انتقال SLR(1) می‌باشد.

و تفاوت فقط در reduce‌ها می‌باشد. در نمودار انتقال LR(0) اگر در حالتی مانند این

شکل وجود داشته باشیم: آنگاه $action[i,a] \leftarrow r_{A\to\alpha}, \forall a \in T \cup \{\$\}$ (یعنی به ازای همه‌ی پایانه‌ها)



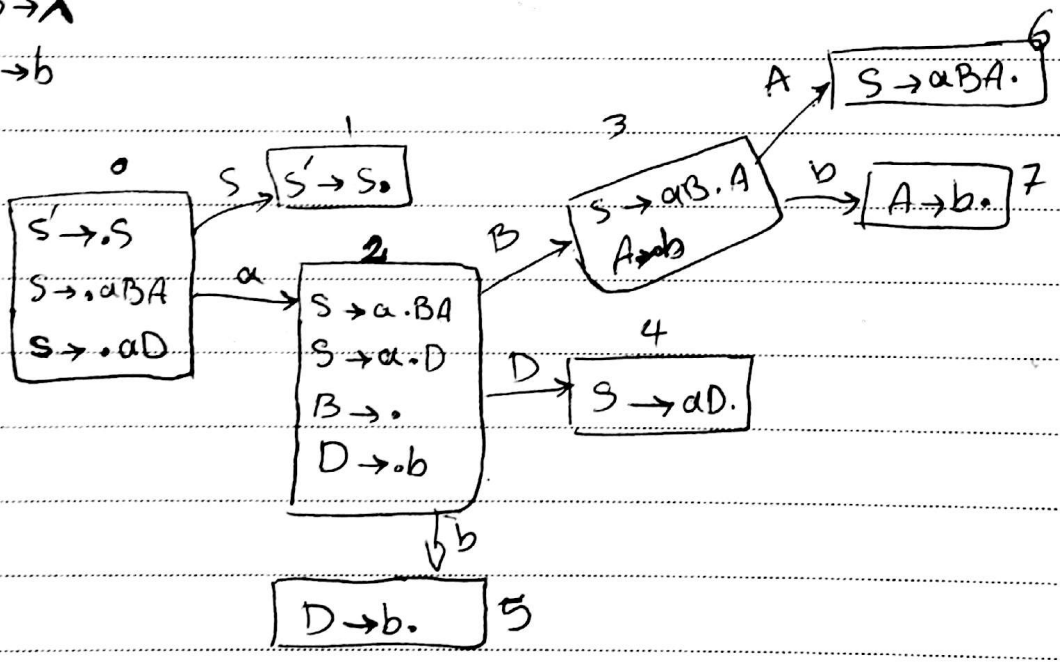
مثال: تمرین 20 ب صفحه 4

$G: S \to aBA \mid aD$

$A \to b$

$B \to \lambda$

$D \to b$



جدول در صفحه‌ی بعد

	a	b	$	S	A	B	D
0	S2			1			
1			acc				
2	r4	S5, r4	r4		3	3	4
3		S7			6		
4	r2	r2	r2				
5	r5	r5	r5				
6	r1	r1	r1				
7	r3	r3	r3				

LR(0) نیست، تداخل S/R دارد.

ساخت جدول تجزیه CLR(1)

تعریف هلم LR(1): هر هلم LR(1) از دو بخش تشکیل شده است: $[A \to \alpha . B, LA]$ هلم LR(0)

↳ $LA \subseteq follow(A)$

LookAhead

* از کتاب مطالعه شود.

Subject:
Year: Month: Day: ( )

ساخت جدول تجزیه LALR(1):

ابتدا باید از روی گراف انتقال CLR(1) به گراف انتقال LALR(1) برسیم.

تعریف هسته (core): به تمام اقلام LR(0) یک حالت، از یک حالت LR(1) هسته گوییم.

ابتدا حالت‌های با هسته‌ی یکسان را پیدا کرده و سپس آن‌ها را یکی‌یکی ادغام می‌کنیم. در حالت ادغام شده هسته‌ها را می‌نویسیم و مجموعه‌ی LA را از اجتماع مجموعه LA حالت‌های قابل ادغام بدست می‌آید.

*پس از ساخت گراف انتقال LALR(1)، مثل CLR(1)، از روی آن جدول تجزیه LALR(1) را بدست می‌آوریم.

*
تمرین: 2, 13, 22, 18, 23, 24, 27