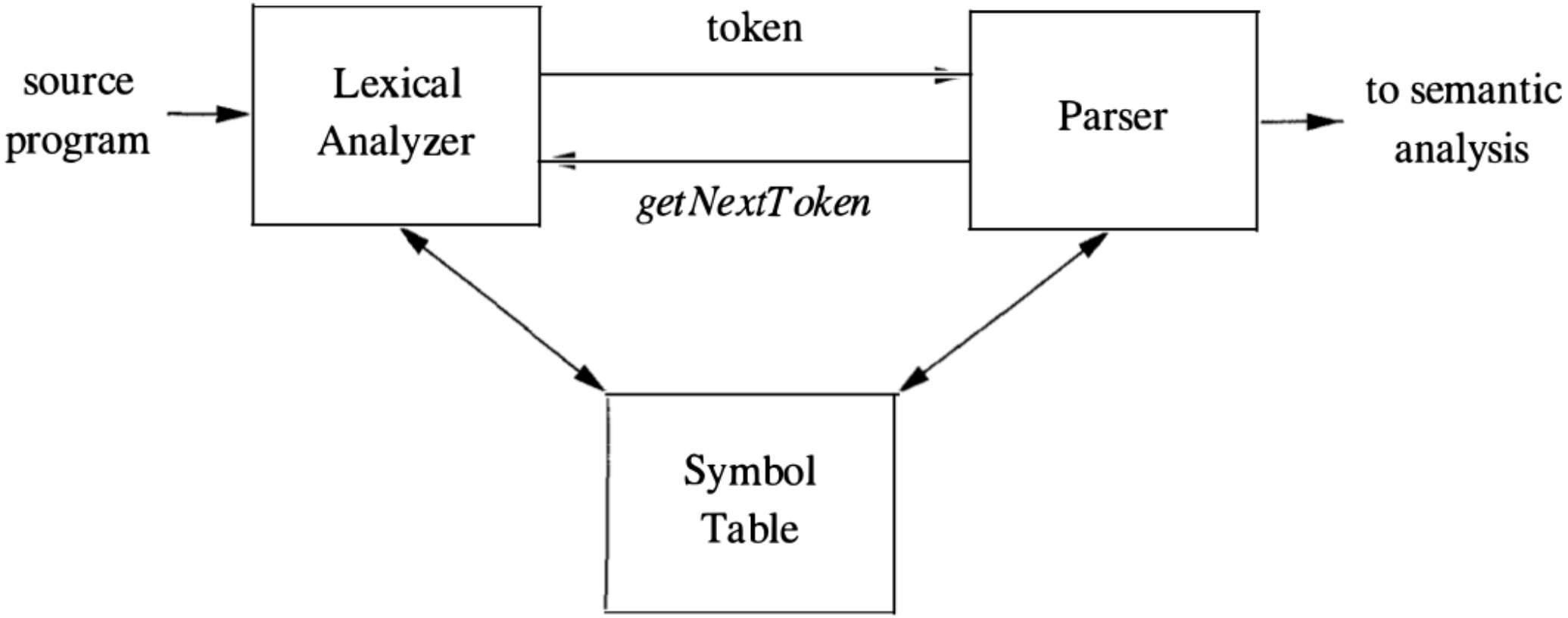
دانشگاه صنعتی شاهرود
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

درس اصول طراحی کامپایلر

تحلیل لغوی
Lexical Analysis

مدرس:
علیرضا تجری

موقعیت تحلیل لغوی در فرآیند کامپایل



عبارت‌های معنادار در زبان‌های برنامه‌نویسی

1) کلمه‌های کلیدی
   - int float if else while
2) علائم
   - ; , ( ) { } [ ]
3) عملگرها
   - > >= ! + – * ** / =
4) شناسه‌ها
   - variableName max ControllerClass
5) ثوابت
   - 15.25 “Hello World” 15
6) توضیحات
   - /*This is a comment*/ // #
7) فضاهای خالی
   - space tab \t new line \n carriage return \r

تحلیلگر معنایی به دنبال کدام یک از موارد بالا است؟

ورودی و خروجی تحلیلگر لغوی

```
int main(){
    return 0;
}
```

کد نوشته شده در فایل

```
int main(){\n\treturn 0;\n}
```

آنچه تحلیلگر لغوی می‌بیند

دنباله توکن‌هایی که برای پارسر ارسال می‌شود

```
<INT> <ID,1> <OPP> <CLP> <OPB> <RETURN>
<NUMBER,0> <SC> <CLB>
```

توکن، الگو و lexeme

- توکن (نشانه، Token): رکوردی شامل یک نام و یک خصوصیت اختیاری

<name, attribute>

- الگو (Pattern): توصیفی از دنباله کاراکترهایی که معادل یک توکن است.

- lexeme: رشته کاراکتری در برنامه ورودی که دارای یک الگو است و توسط تحلیلگر لغوی به عنوان یک توکن شناخته می‌شود.

```
int main(
```

```
<INT> <ID,1> <OPP>
```

مثال توکن، الگو و lexeme

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but ", surrounded by "'s	`"core dumped"`

انواع توکن‌ها

در زبان‌های برنامه نویسی معمولا ۵ دسته توکن وجود دارد:

1) برای هر کلمه‌کلیدی یک توکن وجود دارد. الگوی این توکن، دنباله کاراکترهای کلمه‌کلیدی است.

2) برای عملگرها توکن وجود دارد. می‌توان برای هر عملگر یک توکن داشت و یا برای عملگرهای مشابه یک توکن در نظر گرفت. الگوی این توکن نیز دنباله کاراکتری عملگر است.

3) برای شناسه‌ها (identifier) ــ مانند نام متغییر، نام تابع و نام کلاس ــ یک توکن وجود دارد. الگوی مربوط به این توکن وابسته به زبان برنامه‌نویسی است.

4) یک یا چند توکن نشان دهنده ثابت‌ها (ثابت‌های عددی و رشته‌ای) وجود دارد.

5) توکن‌های مربوط به علائم (, ; : {} []) نیز وجود دارد.

مثال توکن‌ها

1) کلمه‌کلیدی — IF, WHILE, RETURN

2) عملگرها — GT > , LTEQ<=, EQ ==, RELOP

3) شناسه — ID var1 max

4) ثابت‌ها — NUMBER 10.25, STRING_LITERAL “Hi”

5) علائم — OPP (, CLP), OPB {, CLB }

خصوصیات توکن‌ها

- تحلیلگر لغوی در برخی موارد، علاوه بر نام توکن، باید داده‌های بیشتری را همراه آن قرار دهد.
- مثلا هنگامی که یک lexeme از نوع NUMBER است، باید مقدار آن نیز مشخص باشد.
- در مرحله تحلیل گرامر (Syntax analyser)، از نام توکن‌ها استفاده می‌شود.
- در مرحله تحلیل معنا (Semantic analyser)، از خصوصیت توکن‌ها هم استفاده می‌شود.
- برخی از توکن‌ها (مانند شناسه) دارای بیش از یک خصوصیت هستند.
  - مثلا علاوه بر lexeme مربوط به شناسه، باید نوع آن را هم تعیین کرد.
- اطلاعات این توکن‌ها در جدول نمادها (Symbol Table) ذخیره می‌شود.

مثال از دنباله توکن‌های تولید شده

```
E = M * C ** 2
```

<id, pointer to symbol-table entry for E>
<**assign_op**>
<id, pointer to symbol-table entry for M>
<**mult_op**>
<id, pointer to symbol-table entry for C>
<**exp_op**>
<**number**, integer value 2>

الگوی توکن‌ها را چگونه بیان کنیم؟

توصیفی

یک کاراکتر حرفی و بعد از آن
یک یا چند کاراکتر حرفی یا عددی

با عبارت‌های منظم

[a-zA-Z][a-zA-Z0-9]+

عبارت‌های منظم

EXPRESSION	MATCHES	EXAMPLE
c	the one non-operator character c	`a`
$\backslash c$	character c literally	`\*`
`"`s`"`	string s literally	`"**"`
`.`	any character but newline	`a.*b`
`^`	beginning of a line	`^abc`
`$`	end of a line	`abc$`
[s]	any one of the characters in string s	`[abc]`
[^s]	any one character not in string s	`[^abc]`
r*	zero or more strings matching r	`a*`
r+	one or more strings matching r	`a+`
r?	zero or one r	`a?`
$r\{m,n\}$	between m and n occurrences of r	`a[1,5]`
$r_1 r_2$	an r_1 followed by an r_2	`ab`
$r_1 \mid r_2$	an r_1 or an r_2	`a|b`
(r)	same as r	`(a|b)`

مقدمه‌ای بر عبارت‌های منظم

- اولویت عملگرها

- * + ?
- ()
- concat
- |

ad+b

ab?c

ab|cd

a(b|c)d

(ab)|(cd)

(ab*)|(cd+)

a|bc|d+e

نمونه الگو

- یک رقم
- اعداد صحیح
- عدد اعشاری
- نماد علمی
- شناسه
- کلمه کلیدی while
- عملگرهای رابطه‌ای
- عملگر تقسیم
- علامت ;

تعریف منظم

نامگذاری الگوهای ساده برای استفاده در الگوهای پیچیده

$$d1 \Longrightarrow r1$$

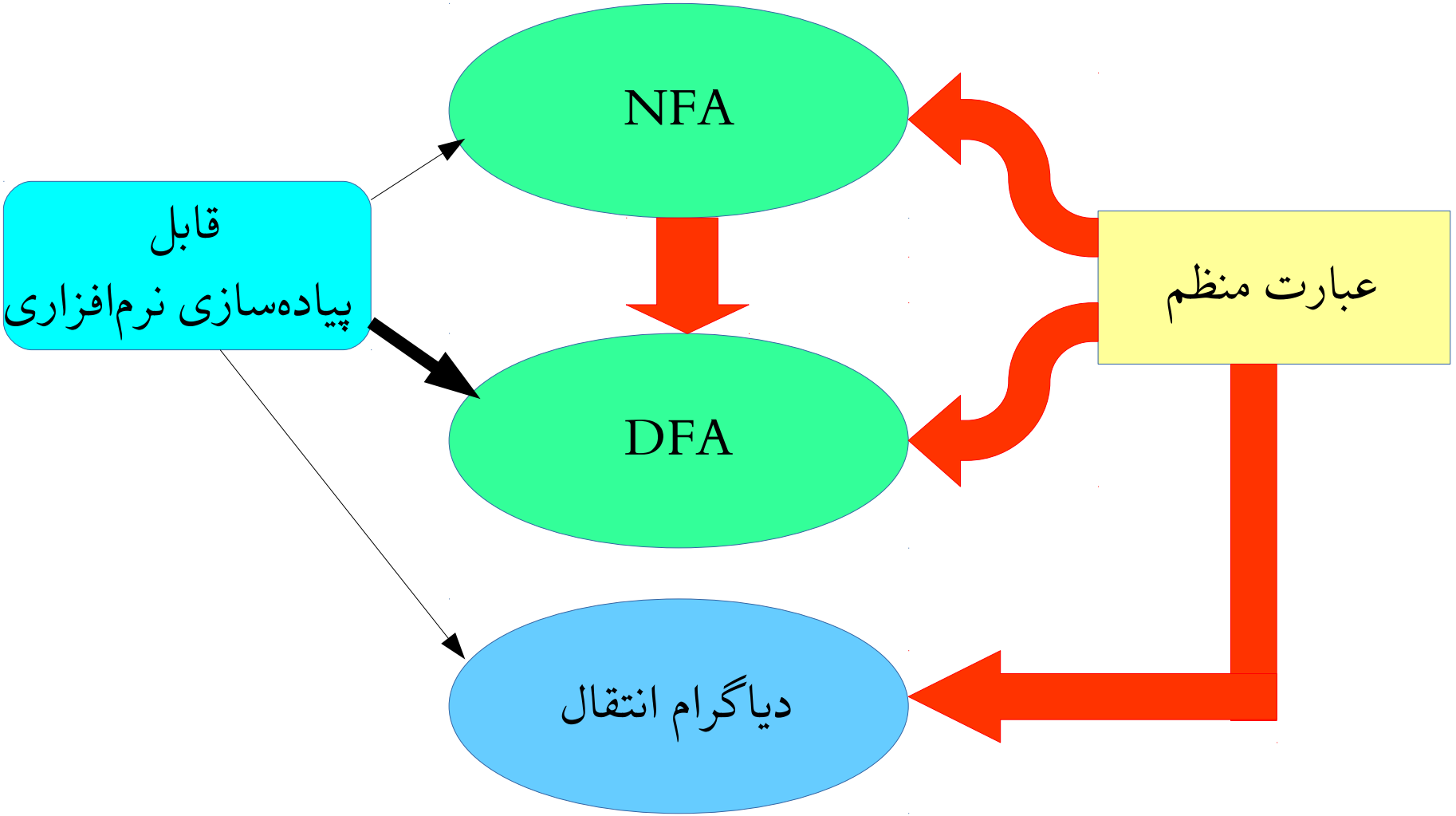
$$d2 \Longrightarrow r2$$

$$dn \Longrightarrow d1\ d2$$

تعریف منظم

$$
\begin{aligned}
\mathit{digit} &\rightarrow \texttt{[0-9]} \\
\mathit{digits} &\rightarrow \mathit{digit}^{+} \\
\mathit{number} &\rightarrow \mathit{digits}\ (\texttt{.}\ \mathit{digits})?\ (\ \texttt{E}\ [\texttt{+-}]?\ \mathit{digits}\ )? \\
\mathit{letter} &\rightarrow \texttt{[A-Za-z]} \\
\mathit{id} &\rightarrow \mathit{letter}\ (\ \mathit{letter} \mid \mathit{digit}\ )^{*} \\
\mathit{if} &\rightarrow \texttt{if} \\
\mathit{then} &\rightarrow \texttt{then} \\
\mathit{else} &\rightarrow \texttt{else} \\
\mathit{relop} &\rightarrow \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{aligned}
$$

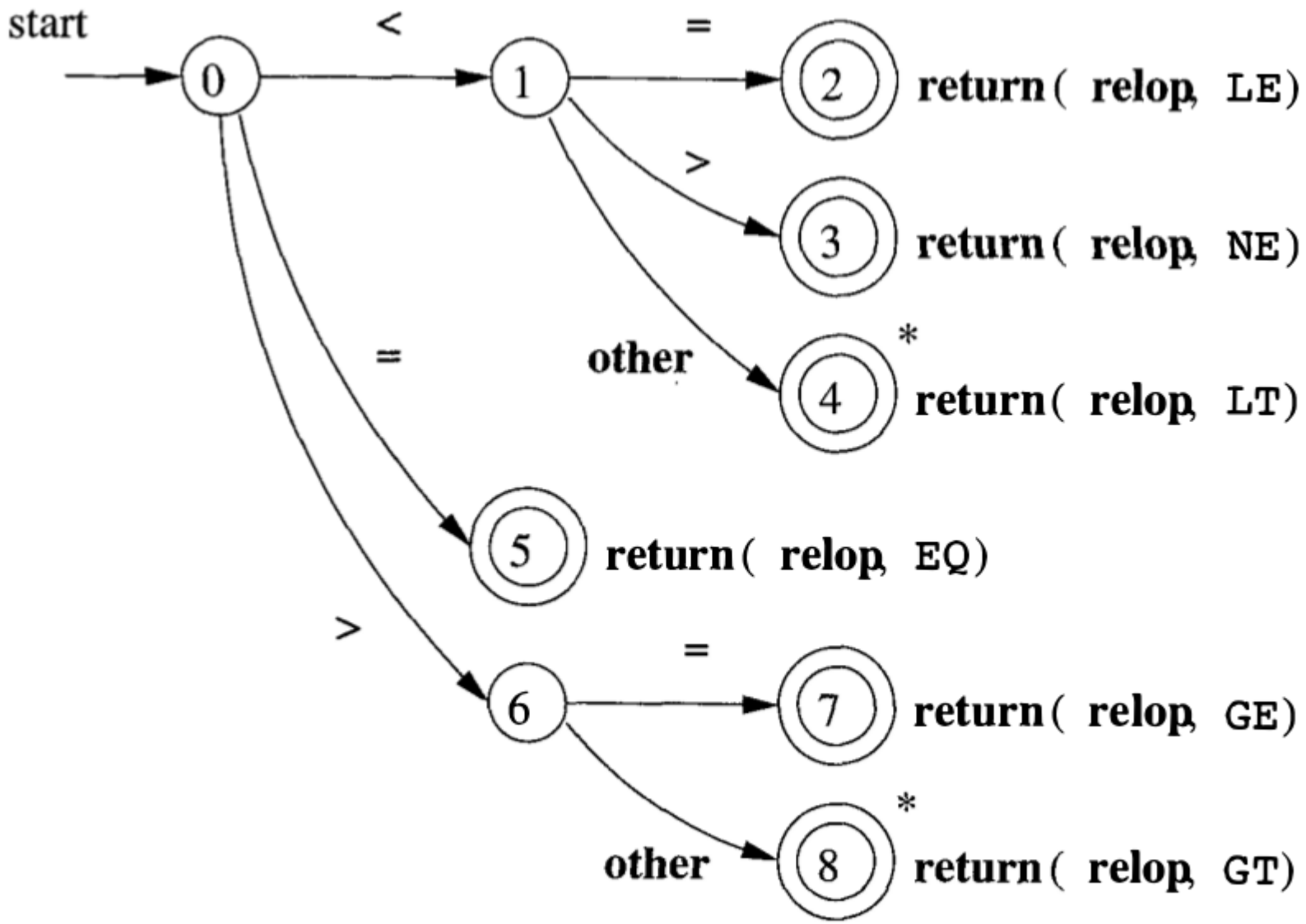
چطور عبارت‌های منظم را پیاده‌سازی کنیم؟



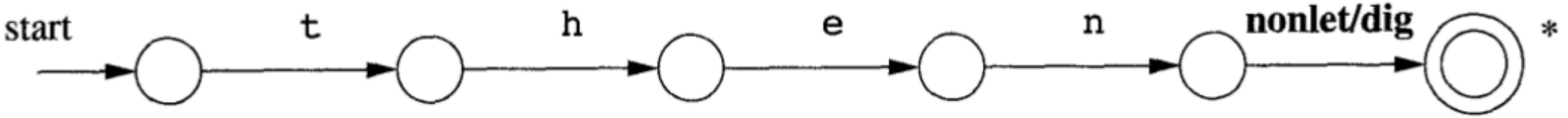
دیاگرام انتقال Transition Diagram

- عبارت‌های منظم را می‌توان به دیاگرام انتقال تبدیل کرد.
- دیاگرام انتقال یک گراف جهت‌دار است.
- به راس‌های دیاگرام، حالت (state) گفته می‌شود. یک حالت شروع نیز وجود دارد.
- یال‌های بین راس‌ها دارای برچسب هستند.
  - یک یا چند کاراکتر
- حالت (state)ها، قطعی (deterministic) هستند.
  - با یک ورودی خاص، از یک حالت فقط به یک حالت دیگر می‌توان رفت.
- برخی از حالت‌ها، پذیرنده (accepting) هستند.
  - حالت پذیرنده نشان‌دهنده کشف یک lexeme است.
- در برخی از حالت‌های پذیرنده، باید ورودی برگردد. در این موارد از * استفاده می‌شود.

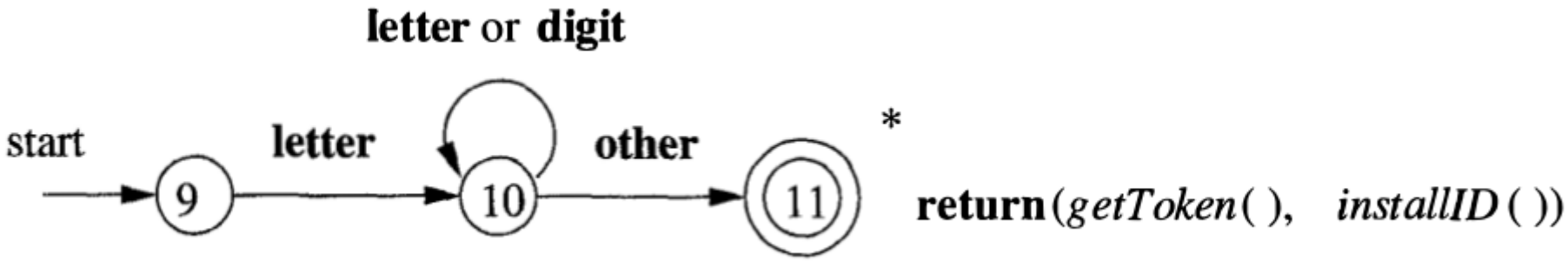
نمونه دیاگرام انتقال



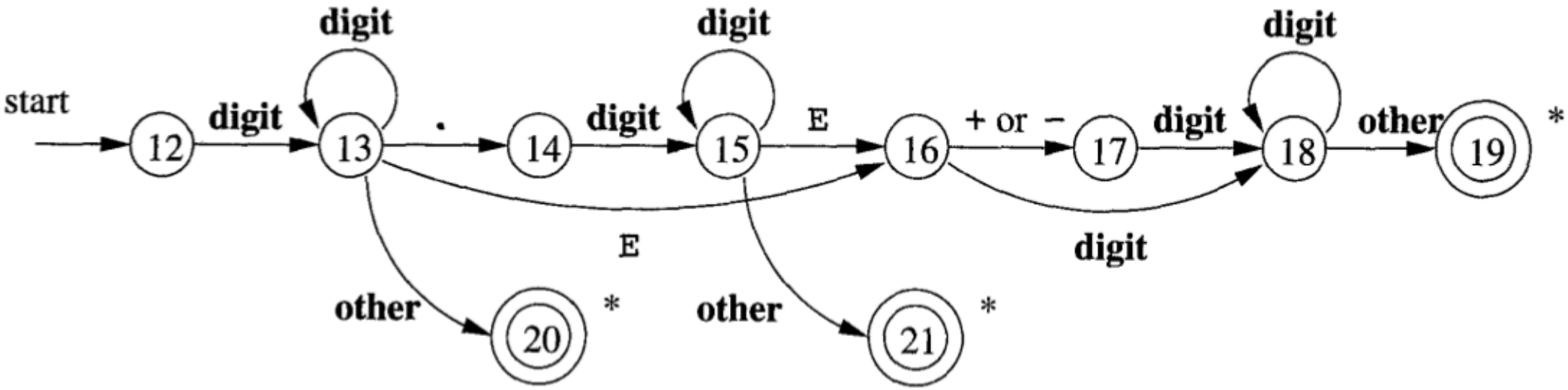
نمونه دیاگرام انتقال



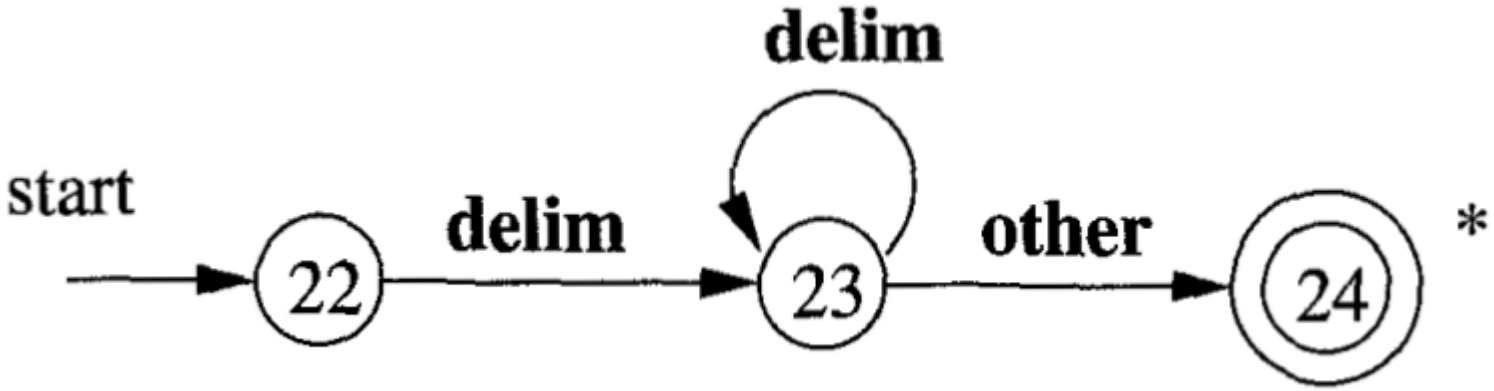
نمونه دیاگرام انتقال



نمونه دیاگرام انتقال



نمونه دیاگرام انتقال



نمونه دیاگرام انتقال

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```