Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Lexical Analysis

- Since the lexical analyzer is the part of the compiler that reads the source text, it may perform certain other tasks besides identification of lexemes
  1. **Stripping out comments and whitespace**
  2. **Correlating error messages generated by the compiler with the source program**
     - For instance, the lexical analyzer may keep track of the number of newline characters seen, so it can associate a line number with each error message

-
تحلیل گر لغوی غیر از شناسایی لغات کارهای دیگری هم انجام میده:
حذف کردن کامنت ها و اسپیس های اضافی
پیام های خطایی که تولید شده رو بخواد مرتبط کنه با برنامه مبدا مثلا کامپایلرها معمولا خطایی که میدن شماره خط رو میدن --> این ها هم معمولا کارهای اولیه اش توی این فاز انجام میشه -->
ساده ترین حالتش این است که برنامه رو بخونه و به ازای هر اینتر یک شماره خط بده به برنامه ینی از یک شروع کنه و شماره خط بده و یک برنامه ای که شماره خط داره تولید کنه و که توی فازهای بعد اگر خطایی به وجود اومد بدونن این خطا مربوط به چه شماره خطی است --> این شماره خط هم توی این فاز انجام می شه

Lexical Analysis

- Sometimes, lexical analyzers are divided into a cascade of two processes:
  1. **Scanning** consists of the simple processes that do not require tokenization of the input, such as:
     - **Deletion of comments**
     - **Compaction of consecutive whitespace characters into one**
  2. **Lexical analysis** produces tokens from the output of the scanner

-
بعضی جاها فاز تحلیل گر لغوی رو به این دوبخش تقسیم می کنن:
1- فاز اسکن کردن --> همون کارهایی میشه که توی اسلاید قبلی گفتیم و از اول تا اخر می ره
2- بعد از اسکن کردن اون قسمت اصلی که همون تحلیل لغوی است از اول تا اخر نمی ره بلکه یک حرف یک حرف لغات رو یکی یکی تشخیص میده و هر یک دونه هم تشخیص داد میده به تحلیل گر نحوی

Lexical Analysis

- What does the lexical analyzer want to do?
- **Example**

```
if (i == j)
  Z = 0;
else
  Z = 1;
```

- The input is just a string of characters:
  - **\tif (i == j)\n\t\tz = 0;\n\telse\n\t\tz = 1;** ورودی تحلیل گر لغوی رشته ای از کاراکترها است
- **Goal:** Partition input string into substrings where the substrings are called **tokens**

--

Tokens

- **What's a Token?**
  - A syntactic category
    - **In English:**
      - **noun, verb, adjective, …**
    - **In a programming language:**
      - **Identifier, Integer, Keyword, Whitespace, …**

- A token class corresponds to a set of strings
  - **Examples**
    - **Identifier**: Strings of letters or digits, starting with a letter
    - **Integer**: A non-empty string of digits
    - **Keyword**: "else" or "if" or "for" or …
    - **Whitespace**: A non-empty sequence of blanks, newlines, and tabs

اینا اسامی token است که می تونه
متفاوت باشه بین زبان های مختلف

- **What are Tokens For?**
  - Classify program substrings according to role
    - An identifier is treated differently than a keyword

Tokens, Patterns, and Lexemes

- **A token** is a pair consisting of a **token name** and **an optional attribute value**
  - The token names are the input symbols that the parser processes
- **A pattern** is a description of the form that the lexemes of a token may take
  - **Example**
    - **In the case of a keyword as a token**, the pattern is the sequence of characters that form the keyword
    - **In the case of an identifier as a token**, the pattern is a more complex structure that is matched by many strings
- **A lexeme** is a sequence of characters in the source program that matches the pattern for a token

-
lexeme: هر لغتی خودش میشه یک lexeme مثلا if

pattern هم اون الگویی است که کامپایلر متوجه میشه که این lexeme مربوط به چه token است ینی token به چه منطبق میشه --> مثلا متغییر position اینکه الگو داشته باشه که بفهمه این مربوط به token Identifier میشه --> قسمت اصلی که باید طراحی بشه توی فاز تحلیل گر لغوی همین patternها است که از روی چه پترنی این رو تشخیص بده

جدول نماد میشه مجموعه tokenها

Tokens, Patterns, and Lexemes

- In many programming languages, the following classes cover most or all of the tokens:
  1. **One token for each keyword**
     - The pattern for a keyword is the same as the keyword itself
  2. **Tokens for the operators**, such as the token comparison
  3. **One token representing all identifiers**
  4. **One or more tokens representing constants**, such as numbers and literal strings
  5. **Tokens for each punctuation symbol**, such as left and right parentheses, comma, and semicolon

-

1- خیلی از زبان های برنامه نویسی برای هر keyword یک توکن در نظر می گیرن --> کلمات کلیدی صفت خاصی غیر از همون اسمشون ندارن و چون اینا صفت خاصی ندارن و فقط اسمشون هست خیلی از زبان ها میان هر کلمه کلیدی رو میگیرن یک توکن ینی مثلا یک توکن داریم به اسم if

2- اپراتورها رو به عنوان توکن در نظر می گیرن

3- برای همه متغییرهای کاربر به اسم id تعریف میشه

4- موارد ثابت هم مثل رشته و اعداد و.. هم توکن هست

5- و علائم هم توکن است مثل پرانتز باز و پرانتز بسته و کاما و...

Tokens, Patterns, and Lexemes

- **Example**

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"`

اینا قسمت اسمشون هست توی توکن

مثال:
توی این مثال هر کلمه کلیدی خودش شده یک توکن است
برای همه عملگرهای که مقایسه ای هستن هم یک توکن گرفته
و..

Attributes for Tokens

- We can assume that tokens have at most one associated attribute, although this attribute may have a structure that combines several pieces of information
- **Example**
  - Information about an **identifier** is kept in the symbol table, including:
    - Its lexeme
    - Its type
    - The location at which it is first found
  - **The appropriate attribute value for an identifier is a pointer to the symbol-table entry for that identifier**

-
صفات:
هر توکن می تونه یکسری صفت داشته باشه --> برای اینکه ساختار جدول نماد حفظ بشه و دو ستونی باشه فقط یک اسم می گیرن و برای صفتش هم یک پوینتر میگیرن به یک لیستی که صفات توی اون لیست ذخیره میشه یا اینکه می تونه شماره سطر باشه توی جدول نماد
و صفت ها می تونه متنوع باشه توی کامپایلرهای مختلف --> مثلا توکن identifier --> اسم lexeme است و تایپش int و مکانی که اولین بار توی برنامه اومده هم میشه ذخیره کرد برای خطایابی کمک می کنه --> این صفت ها توی جدول نماد ذخیره میشه

Attributes for Tokens

- **Example**
  - The token names and associated attribute values for the Fortran statement: 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>

  - *Note that in certain pairs, especially operators, punctuation, and keywords,* ***there is no need for an attribute value***

-
مثال: توکن هایی که برای این عبارت داریم عبارتند از:
توی این مثال صفتش رو به صورت پوینتر نوشته
توی این مثال برای هر اپراتور به صورت جداگانه یک توکن در نظر گرفته --> هر کدوم از این عملگرها رو توکن هاشو متمایز از هم گرفته که نخواد برای هر کدوم صفت در نظر بگیره چون اگر یکی بود باید مثل id براش صفت هم در نظر میگرفت

Specification of Tokens

- **Regular expressions** are an important notation for specifying lexeme patterns
- **Strings and Languages**
  - An **alphabet** is any finite set of symbols
    - Letters, digits, and punctuation
  - A **string** over an alphabet is a finite sequence of symbols drawn from that alphabet
    - The terms "sentence" and "word" are often used as synonyms for "string"
  - A **language** is any countable set of strings over some fixed alphabet

-
دوره درس نظریه:
عبارات منظم برای اینکه بتونیم پترن ها رو تعریف بکنیم استفاده میشن و خیلی کاربرد داره مثلا یکی از کاربردهاش میشه تحلیل گر لغوی که از عبارات منظم استفاده می کنن برای تشخیص lexemeها
پس کلا تحلیل گر لغوی میشه یک مجموعه ای از NFA , DFA ها

الفبا: یک مجموعه متناهی از کاراکترهاست که این کاراکترها می تونه حرف باشه رقم باشه یا ..
رشته: یک دنباله ای متناهی روی الفبا ینی یک الفبا داریم و روش می تونیم بی نهایت رشته داشته باشیم
زبان: یک مجموعه متناهی از رشته ها
مثلا الفبا هست a , b و رشته میشه هر ترکیبی از این دوتا ینی aa , ab , ba, bb و زبان هم هست مجموعه ای از رشته ها

Operations on Languages

- In lexical analysis, the most important operations on languages are **union**, **concatenation**, and **closure**

Operation	Definition and Notation
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

- **Example**
  - Let L be the set of letters {A, B, …, Z, a, b, …, z} and let D be the set of digits {0, 1, …, 9}
    - $L \cup D$, LD, L^4, L^*, $L(L \cup D)^*$, D^+

-
عملیات اصلی که روی زبان ها انجام میشه و توی تحلیل گر لغوی هم خیلی ازش استفاده میشه عبارتند از:
اجتماع یا union - الحاق یا concatenation - بستار یا closure

Regular Expressions

- **Basis**
  - ϵ is a regular expression, and $L(\epsilon)$ is $\{\epsilon\}$
  - If a is a symbol in $\sum$, then $\boldsymbol{a}$ is a regular expression, and $L(\boldsymbol{a}) = \{a\}$
- **Induction**
  1. $(r)|(s)$ is a regular expression denoting the language $L(r) \cup L(s)$
  2. $(r)(s)$ is a regular expression denoting the language $L(r)L(s)$
  3. $(r)^*$ is a regular expression denoting $(L(r))^*$
  4. (r) is a regular expression denoting $L(r)$

-
عبارات منظم و DFA , NFA معمولا به صورت بازگشتی تعریف می شن
قسمت Basis اون حالت پایه ای است که برای تعریف عبارات منظم استفاده میشه --> توی حالت پایه میگیم یا رشتمون رشته تهی است که با اپسیلون نشونش میدیم (اگر اپسیلون نشون دهنده رشته تهی باشه خود این اپسیلون می تونه یک زبان رو تشکیل بده) - هر تک حرفی هم از الفبامون که با سیگما نشونش داده می تونه یک زبان ساده باشه که شامل اون حرف فقط باشه --> این حالت پایه است
هر زبانی که ساخته میشه روی یک الفبا از ترکیب همین دوتا حالت پایه بالا است

اگر دوتا عبارت منظم داشته باشیم --> اجتماعشون و الحاقشون و بستارشون و خود اون عبارت توی پرانتز هم میشه منظم

Regular Expressions

- **Precedences**
  - The unary operator * has highest precedence
  - Concatenation has second highest precedence
  - | has lowest precedence
- **Example**
  - $\sum = \{a, b\}$
  - $(a|b)(a|b) \rightarrow \{aa, ab, ba, bb\}$
- A language that can be defined by a regular expression is called a regular set

-
اولویت ها:
بستار اولویتش از همه بیشتر است و بعد الحاق و بعد اجتماع

Regular Expressions

- Algebraic laws for regular expressions

Law
$r\|s = s\|r$
$r\|(s\|t) = (r\|s)\|t$
$r(st) = (rs)t$
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$
$\epsilon r = r\epsilon = r$
$r^* = (r\|\epsilon)^*$
$r^{**} = r^*$

- قوانین جبری برای عبارات منظم

Regular Expressions

- **Example**
  - A regular definition for the language of C identifiers

این id همون اسم توکن است که الان این داره میگه که متغییر نمی تونه با عدد شروع بشه

$$
\begin{array}{rcl}
letter\_ & \rightarrow & \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\
digit & \rightarrow & \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
id & \rightarrow & letter\_ \; (\; letter\_ \mid digit \;)^*
\end{array}
$$

  - A regular definition for unsigned numbers (such as 5280, 0.034, 6.36E4, or 1.89E-4)

$$
\begin{array}{rcl}
digit & \rightarrow & \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
digits & \rightarrow & digit \; digit^* \\
optionalFraction & \rightarrow & \texttt{.} \; digits \mid \epsilon \\
optionalExponent & \rightarrow & (\; \texttt{E} \; (\; \texttt{+} \mid \texttt{-} \mid \epsilon \;) \; digits \;) \mid \epsilon \\
number & \rightarrow & digits \; optionalFraction \; optionalExponent
\end{array}
$$

برای داشتن نماد علمی E اجباری است

-

مثال دومی:
فرض کنید می خوایم هر عددی رو مثلا عدد صحیح و عدد اعشاری و اعشاری با نماد علمی تعریف کنیم

Regular Expressions

- **Example**
  - Describe the languages denoted by the following regular expressions:

a) $\mathbf{a}(\mathbf{a}|\mathbf{b})^*\mathbf{a}$.

b) $((\epsilon|\mathbf{a})\mathbf{b}^*)^*$.

c) $(\mathbf{a}|\mathbf{b})^*\mathbf{a}(\mathbf{a}|\mathbf{b})(\mathbf{a}|\mathbf{b})$.

d) $\mathbf{a}^*\mathbf{b}\mathbf{a}^*\mathbf{b}\mathbf{a}^*\mathbf{b}\mathbf{a}^*$.

e) $(\mathbf{aa}|\mathbf{bb})^*((\mathbf{ab}|\mathbf{ba})(\mathbf{aa}|\mathbf{bb})^*(\mathbf{ab}|\mathbf{ba})(\mathbf{aa}|\mathbf{bb})^*)^*$.

- مثال

زبان هایی که با عبارات منظم زیر مشخص می شوند را توصیف کنید:

Recognition of Tokens

- **Example**
  - A grammar for branching statements (for Pascal language)

stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *stmt* **else** *stmt*
	|	ϵ
expr	→	*term* **relop** *term*
	|	*term*
term	→	**id**
	|	**number**

-
این یک مثال است برای عبارات شرطی توی زبان پاسکال

Recognition of Tokens

- **Example**
  - Patterns for tokens

digit	→	`[0-9]`
digits	→	*digit*$^+$
number	→	*digits* (`.` *digits*)? (`E` `[+-]`? *digits*)?
letter	→	`[A-Za-z]`
id	→	*letter* (*letter* | *digit*)*
if	→	`if`
then	→	`then`
else	→	`else`
relop	→	`<` | `>` | `<=` | `>=` | `=` | `<>`

  - Assign the lexical analyzer the job of stripping out white-space

$$ws \rightarrow (\ \textbf{blank} \mid \textbf{tab} \mid \textbf{newline}\ )^+$$

Recognition of Tokens

- **Example**
  - Tokens, their patterns, and attribute values

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
`<`	**relop**	`LT`
`<=`	**relop**	`LE`
`=`	**relop**	`EQ`
`<>`	**relop**	`NE`
`>`	**relop**	`GT`
`>=`	**relop**	`GE`

--