CS4120/4121

Introduction to Compilers

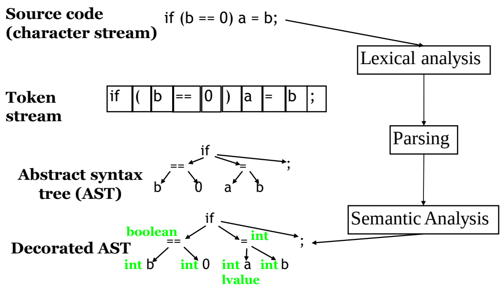
Ross

Lecture 2: Lexical Analysis

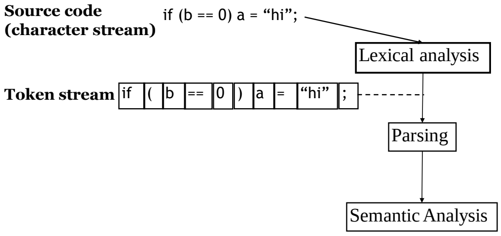
Administration

- HW1 out later today – due next Monday.

Compilation Recap



First step: lexical analysis



Tokens

- Identifiers: x y11 elsex _i00
- Keywords: if else while break
- Integers: 2 1000 -500 5L
- Floating point: 2.0 0.00020 .02 1. 1e5 0.e-10
- Symbols: + * { } ++ < << [ ] >=
- Strings: "x" "He said, \"Are you?\""
- Comments: /** don't change this **/

Ad-hoc lexer

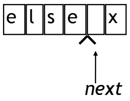
- Hand-write code to generate tokens
- How to read identifier tokens?

```
Token readIdentifier( ) {
  String id = "";
  while (true) {
   char c = input.read();
   if (!identifierChar(c))
          return new Token(ID, id, lineNumber);
   id = id + String(c);
  }
}
```

Look-ahead character

- Scan text one character at a time
- Use look-ahead character (**next**) to determine what kind of token to read *and* when the current token ends

```
char next;
…
while (identifierChar(next)) {
    id = id + String(next);
    next = input.read ();
}
```



Ad-hoc lexer: top-level loop

```
class Lexer {
    InputStream s;
    char next;
    Lexer(InputStream s_) { s = s_; next = s.read(); }
    Token nextToken( ) {
        if (identifierChar(next))
                return readIdentifier();
        if (numericChar(next))
                return readNumber();
        if (next == '\"') return readStringConst();
        …
    }
}
```

Preloading **next**.

Alternative: define input streams that support lookahead automatically

Problems

- Don't know what kind of token we are going to read from seeing first character
  - if token begins with "i'' is it an identifier or "if"?
  - if token begins with "2" is it an integer constant?
  - interleaved tokenizer code is hard to write correctly, harder to maintain
- A more principled approach: *lexer generator* that generates efficient tokenizer automatically (e.g., lex, Jlex, ANTLR) from a lexical specification.

Lexer Generator

- Input
  - Description of the tokens
  - Prioritization of the tokens
  - Actions for the tokens
- Output
  - A lexer
    - Matching the specification
    - Efficient (linear time)

Issues

- How to describe tokens unambiguously

  2.e0   20.e-01   2.0000

  "" "x"   "\\"   "\"\'"
- How to break text up into tokens

  if (x == 0) a = x<<1;

  if (x == 0) a = x<1;
- How to tokenize efficiently
  - tokens may have similar prefixes
  - want to look at each character O(1) times

How to Describe Tokens

- Programming-language tokens can (often) be described using **regular expressions**
- Regular expression R describes a set of strings L(R):

  L(R) is the "language" defined by R
  - L(**abc**) = { **abc** }
  - L(**hello|goodbye**) = {**hello**, **goodbye**}
  - L(**[1-9][0-9]***) = all positive integer constants
  - L(X(Y|Z)) = L(XY|XZ) = L(XY) U L(XZ)
- Idea: define each kind of token using REs

Regular-Expression Notation

a an ordinary character stands for itself

ε the empty string

R|S any string from either L(R) or L(S):
L(R|S)=L(R)U L(S)

RS string from L(R) followed by one from L(S):
L(RS) = {rs | r∈L(R) ^ s∈L(S)}

R* zero or more strings from L(R), concatenated
ε|R|RR|RRR|RRRR|… (“Kleene star”)

Examples

Regular Expression R	Strings in L(R)
a	“a”
ab	“ab”
a | **b**	“a” “b”
ε	“”
(**ab**)*	“” “ab” “abab” …
(**a**|ε)**b**	“ab” “b” (=**a?b**)

Convenient RE Shorthand

R+ one or more strings from L(R): = R(R*)

R? an optional R: = (R|ε)

[**abce**] one of the listed characters: (**a**|**b**|**c**|**e**)

[**a-z**] one char from the range: (**a**|**b**|**c**|**d**|**e**|**…**)

[^**ab**] anything but one of the listed chars

[^**a-z**] one character **not** from the range
(~[ab] and ~[a-z] in ANTLR)

R{n} n repetitions of R (RRRR…)

\x0A ASCII 10 (newline)

\n also newline

More Examples (JFlex)

Regular Expression	Strings in L(R)
digit = [**0-9**]	“0” “1” “2” “3” …
posint = {*digit*}+	“8” “412” …
int = -? {*posint*}	“-42” “1024” …
real = {*int*} (. *posint*)?	“-1.56” “12” “1.0”
= (-| ε)(0|…|9)(0|...|9)*(ε | (. (0|...|9)(0|...|9)*))	
[**a-zA-Z_**][**a-zA-Z0-9_**]*	C identifiers

- Lexer generators support abbreviations – cannot be recursive. Forbidden: *foo* **= a**{*foo*}| ε
  - Actually, ANTLR v4 can! And you’ll need it

Zero-width assertions

- Not strictly regular expressions…
- Not supported by all lexer generators.

^R matches R if preceded by newline

R$ matches R if followed by newline

\b match a word boundary (Perl)

\A match beginning of input (Perl)

R_1/R_2 matches R_1 if followed by
something matching R_2 (lex)

How to break up text

elsex = 0;

1 else x = 0

2 elsex = 0

- REs alone not enough: need rule for choosing
- Most languages: **longest matching token** wins – even if a shorter token is only way to parse tokens.
  - Exception: early FORTRAN (totally whitespace-insensitive)
  - Ties in length resolved by prioritizing tokens
- RE’s + priorities + longest-matching token rule = lexer definition

Lexer-Generator Spec

- Input to lexer generator:
  - list of regular expressions in priority order
  - associated *action* for each RE (generates appropriate kind of token, other bookkeeping)
- Output:
  - program that reads an input stream and breaks it up into tokens according to the REs. (Or reports lexical error -- *"Unexpected character"*)

Example: ANTLR v4

```
lexer grammar XiLexer;

ELSE : 'else';
ID : ([a-zA-Z]) ([a-zA-Z_0-9]|'\'')*;
SLASH : '/';
WS : [ \t\r\n]+ -> skip;
COMMENT : '//' .*? [\r\n] -> skip;
```

Lexer States

- Most lexer generators allow conditioning on lexer state. Helps with long tokens (strings, comments):

```
"/*"       { yybegin(COMMENT); }
<COMMENT> {
  "*/"   { yybegin(YYINITIAL); }
  .|\n      { /* ignore */ }
}
```

Summary

- Lexical analyzer converts a text stream to tokens
- Ad-hoc lexers hard to get right, maintain
- For most languages, legal tokens conveniently, precisely defined using regular expressions
- Lexer generators generate lexer code automatically from token RE's, precedence
- Next lecture: how lexer generators work

Groups

- If you don't have a full group lined up, hang around and talk to prospective group members
- Send mail to cs4120-l if you still cannot make a full group (can also post to Piazza)