

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC

<http://courses.engr.illinois.edu/cs421>

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

10/25/12

1

General Input

```
{ header }  
let ident = regexp ...  
rule entrypoint [arg1... argn] = parse  
    regexp { action }  
    | ...  
    | regexp { action }  
and entrypoint [arg1... argn] =  
    parse ...and ...  
{ trailer }
```

10/25/12

2

Ocamllex Input

- *header* and *trailer* contain arbitrary ocaml code put at top and bottom of *<filename>.ml*
- `let ident = regexp ...` Introduces *ident* for use in later regular expressions

10/25/12

3

Ocamllex Input

- *<filename>.ml* contains one lexing function per *entrypoint*
 - Name of function is name given for *entrypoint*
 - Each entry point becomes an Ocaml function that takes $n+1$ arguments, the extra implicit last argument being of type `Lexing.lexbuf`
- *arg1... argn* are for use in *action*

10/25/12

4

Ocamllex Regular Expression

- Single quoted characters for letters: `'a'`
- `_`: (underscore) matches any letter
- `Eof`: special "end_of_file" marker
- Concatenation same as usual
- `"string"`: concatenation of sequence of characters
- e_1 / e_2 : choice - what was $e_1 \vee e_2$

10/25/12

5

Ocamllex Regular Expression

- $[c_1 - c_2]$: choice of any character between first and second inclusive, as determined by character codes
- $[^c_1 - c_2]$: choice of any character NOT in set
- e^* : same as before
- e^+ : same as $e e^*$
- $e?$: option - was $e_1 \vee \epsilon$

10/25/12

6

Ocamllex Regular Expression

- $e_1 \# e_2$: the characters in e_1 but not in e_2 ; e_1 and e_2 must describe just sets of characters
- *ident*: abbreviation for earlier reg exp in
let *ident* = *regexp*
- e_1 as *id*: binds the result of e_1 to *id* to be used in the associated *action*

10/25/12

7

Ocamllex Manual

- More details can be found at

<http://caml.inria.fr/pub/docs/manual-ocaml/manual026.html>

10/25/12

8

Example : test.mll

```
{ type result = Int of int | Float of float |  
  String of string }  
let digit = ['0'-'9']  
let digits = digit +  
let lower_case = ['a'-'z']  
let upper_case = ['A'-'Z']  
let letter = upper_case | lower_case  
let letters = letter +
```

10/25/12

9

Example : test.mll

```
rule main = parse  
  (digits)'.digits as f { Float (float_of_string f) }  
  | digits as n          { Int (int_of_string n) }  
  | letters as s          { String s }  
  | _ { main lexbuf }  
{ let newlexbuf = (Lexing.from_channel stdin) in  
  print_string "Ready to lex."  
  print_newline ();  
  main newlexbuf }
```

10/25/12

10

Example

```
# #use "test.ml";;  
...  
val main : Lexing.lexbuf -> result = <fun>  
val __ocaml_lex_main_rec : Lexing.lexbuf -> int ->  
  result = <fun>  
Ready to lex.  
hi there 234 5.2  
- : result = String "hi"  
What happened to the rest?!?
```

10/25/12

11

Example

```
# let b = Lexing.from_channel stdin;;  
# main b;;  
hi 673 there  
- : result = String "hi"  
# main b;;  
- : result = Int 673  
# main b;;  
- : result = String "there"
```

10/25/12

12

Problem

- How to get lexer to look at more than the first token at one time?
- Answer: *action* has to tell it to -- recursive calls
- Side Benefit: can add "state" into lexing
- Note: already used this with the `_` case

10/25/12

13

Example

```
rule main = parse
  (digits) '.' digits as f { Float (float_of_string f) :: main lexbuf }
| digits as n              { Int (int_of_string n) ::
  main lexbuf }
| letters as s             { String s :: main
  lexbuf }
| eof                      { [] }
| _                        { main lexbuf }
```

10/25/12

14

Example Results

Ready to lex.

hi there 234 5.2

- : result list = [String "hi"; String "there"; Int 234; Float 5.2]

#

Used Ctrl-d to send the end-of-file signal

10/25/12

15

Dealing with comments

First Attempt

```
let open_comment = "("
let close_comment = ")"
rule main = parse
  (digits) '.' digits as f { Float (float_of_string f) :: main lexbuf }
| digits as n              { Int (int_of_string n) ::
  main lexbuf }
| letters as s             { String s :: main lexbuf }
```

10/25/12

16

Dealing with comments

```
| open_comment      { comment lexbuf }
| eof               { [] }
| _ { main lexbuf }
and comment = parse
  close_comment     { main lexbuf }
| _                { comment lexbuf }
```

10/25/12

17

Dealing with nested comments

```
rule main = parse ...
| open_comment      { comment 1 lexbuf }
| eof               { [] }
| _ { main lexbuf }
and comment depth = parse
  open_comment      { comment (depth+1)
  lexbuf }
| close_comment     { if depth = 1
                      then main lexbuf
                      else comment (depth - 1) lexbuf }
| _                 { comment depth lexbuf }
```

10/25/12

18

Dealing with nested comments

```
rule main = parse
  (digits) '.' digits as f { Float (float_of_string f) ::
    main lexbuf }
| digits as n      { Int (int_of_string n) :: main
  lexbuf }
| letters as s     { String s :: main lexbuf }
| open_comment     { (comment 1 lexbuf }
| eof              { [] }
| _ { main lexbuf }
```

10/25/12

19

Dealing with nested comments

```
and comment depth = parse
  open_comment      { comment (depth+1) lexbuf }
| close_comment     { if depth = 1
                      then main lexbuf
                      else comment (depth - 1) lexbuf }
| _                 { comment depth lexbuf }
```

10/25/12

20

Types of Formal Language Descriptions

- Regular expressions, regular grammars
- Context-free grammars, BNF grammars, syntax diagrams
- Finite state automata
- Whole family more of grammars and automata – covered in automata theory

10/25/12

21

Sample Grammar

- Language: Parenthesized sums of 0's and 1's
- $\langle \text{Sum} \rangle ::= 0$
- $\langle \text{Sum} \rangle ::= 1$
- $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
- $\langle \text{Sum} \rangle ::= (\langle \text{Sum} \rangle)$

10/25/12

22

BNF Grammars

- Start with a set of characters, **a,b,c,...**
 - We call these *terminals*
- Add a set of different characters, **X,Y,Z,**
...
 - We call these *nonterminals*
- One special nonterminal **S** called *start symbol*

10/25/12

23

BNF Grammars

- BNF rules (aka *productions*) have form
 $\mathbf{X} ::= y$
where **X** is any nonterminal and *y* is a string of terminals and nonterminals
- BNF *grammar* is a set of BNF rules such that every nonterminal appears on the left of some rule

10/25/12

24

Sample Grammar

- Terminals: 0 1 + ()
- Nonterminals: <Sum>
- Start symbol = <Sum>
- $\langle \text{Sum} \rangle ::= 0$
- $\langle \text{Sum} \rangle ::= 1$
- $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
- $\langle \text{Sum} \rangle ::= (\langle \text{Sum} \rangle)$
- Can be abbreviated as
 $\langle \text{Sum} \rangle ::= 0 \mid 1$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \mid (\langle \text{Sum} \rangle)$

10/25/12

25

BNF Derivations

- Given rules

$$\mathbf{X} ::= y\mathbf{Z}w \text{ and } \mathbf{Z} ::= v$$

we may replace \mathbf{Z} by v to say

$$\mathbf{X} \Rightarrow y\mathbf{Z}w \Rightarrow yvw$$

- Sequence of such replacements called *derivation*
- Derivation called *right-most* if always replace the right-most non-terminal

10/25/12

26

BNF Derivations

- Start with the start symbol:

$\langle \text{Sum} \rangle \Rightarrow$

10/25/12

27

BNF Derivations

- Pick a non-terminal

$\langle \text{Sum} \rangle \Rightarrow$

10/25/12

28

BNF Derivations

- Pick a rule and substitute:
 - $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
- $\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

10/25/12

29

BNF Derivations

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

10/25/12

30

BNF Derivations

- Pick a rule and substitute:

■ $\langle \text{Sum} \rangle ::= (\langle \text{Sum} \rangle)$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$

10/25/12

31

BNF Derivations

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$

10/25/12

32

BNF Derivations

- Pick a rule and substitute:

■ $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$

10/25/12

33

BNF Derivations

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$

10/25/12

34

BNF Derivations

- Pick a rule and substitute:

■ $\langle \text{Sum} \rangle ::= 1$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$

10/25/12

35

BNF Derivations

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$

10/25/12

36

BNF Derivations

- Pick a rule and substitute:

■ $\langle \text{Sum} \rangle ::= 0$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + 0$

10/25/12

37

BNF Derivations

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + 0$

10/25/12

38

BNF Derivations

- Pick a rule and substitute

■ $\langle \text{Sum} \rangle ::= 0$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) 0$
 $\Rightarrow (0 + 1) + 0$

10/25/12

39

BNF Derivations

- $(0 + 1) + 0$ is generated by grammar

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + \langle \text{Sum} \rangle$
 $\Rightarrow (\langle \text{Sum} \rangle + 1) + 0$
 $\Rightarrow (0 + 1) + 0$

10/25/12

40

$\langle \text{Sum} \rangle ::= 0 \mid 1 \mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \mid (\langle \text{Sum} \rangle)$

$\langle \text{Sum} \rangle \Rightarrow$

10/25/12

41

BNF Semantics

- The meaning of a BNF grammar is the set of all strings consisting only of terminals that can be derived from the Start symbol

10/25/12

42

Extended BNF Grammars

- Alternatives: allow rules of form $X ::= y/z$
 - Abbreviates $X ::= y, X ::= z$
- Options: $X ::= y[v]z$
 - Abbreviates $X ::= yvz, X ::= yz$
- Repetition: $X ::= y\{v\}^*z$
 - Can be eliminated by adding new nonterminal V and rules $X ::= yz, X ::= yVz, V ::= v, V ::= W$

10/25/12

43

Regular Grammars

- Subclass of BNF
- Only rules of form $\langle \text{nonterminal} \rangle ::= \langle \text{terminal} \rangle \langle \text{nonterminal} \rangle$ or $\langle \text{nonterminal} \rangle ::= \langle \text{terminal} \rangle$ or $\langle \text{nonterminal} \rangle ::= \epsilon$
- Defines same class of languages as regular expressions
- Important for writing lexers (programs that convert strings of characters into strings of tokens)

10/25/12

44

Example

- Regular grammar:
 - $\langle \text{Balanced} \rangle ::= \epsilon$
 - $\langle \text{Balanced} \rangle ::= 0 \langle \text{OneAndMore} \rangle$
 - $\langle \text{Balanced} \rangle ::= 1 \langle \text{ZeroAndMore} \rangle$
 - $\langle \text{OneAndMore} \rangle ::= 1 \langle \text{Balanced} \rangle$
 - $\langle \text{ZeroAndMore} \rangle ::= 0 \langle \text{Balanced} \rangle$
- Generates even length strings where every initial substring of even length has same number of 0's as 1's

10/25/12

45

Parse Trees

- Graphical representation of derivation
- Each node labeled with either non-terminal or terminal
- If node is labeled with a terminal, then it is a leaf (no sub-trees)
- If node is labeled with a non-terminal, then it has one branch for each character in the right-hand side of rule used to substitute for it

10/25/12

46

Example

- Consider grammar:
 - $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle$
 - $\quad \quad \quad | \langle \text{factor} \rangle + \langle \text{factor} \rangle$
 - $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle$
 - $\quad \quad \quad | \langle \text{bin} \rangle * \langle \text{exp} \rangle$
 - $\langle \text{bin} \rangle ::= 0 \mid 1$
- Problem: Build parse tree for $1 * 1 + 0$ as an $\langle \text{exp} \rangle$

10/25/12

47

Example cont.

- $1 * 1 + 0: \langle \text{exp} \rangle$

$\langle \text{exp} \rangle$ is the start symbol for this parse tree

10/25/12

48

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$

Use rule: $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle$

10/25/12

49

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$
 $\swarrow \quad \downarrow \quad \searrow$
 $\langle \text{bin} \rangle \quad * \quad \langle \text{exp} \rangle$

Use rule: $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle * \langle \text{exp} \rangle$

10/25/12

50

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$
 $\swarrow \quad \downarrow \quad \searrow$
 $\langle \text{bin} \rangle \quad * \quad \langle \text{exp} \rangle$
 $\downarrow \quad \swarrow \quad \downarrow \quad \searrow$
1 $\langle \text{factor} \rangle$ + $\langle \text{factor} \rangle$

Use rules: $\langle \text{bin} \rangle ::= 1$ and
 $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle + \langle \text{factor} \rangle$

10/25/12

51

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$
 $\swarrow \quad \downarrow \quad \searrow$
 $\langle \text{bin} \rangle \quad * \quad \langle \text{exp} \rangle$
 $\downarrow \quad \swarrow \quad \downarrow \quad \searrow$
1 $\langle \text{factor} \rangle$ + $\langle \text{factor} \rangle$
 $\downarrow \quad \downarrow$
 $\langle \text{bin} \rangle \quad \langle \text{bin} \rangle$

Use rule: $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle$

10/25/12

52

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$
 $\swarrow \quad \downarrow \quad \searrow$
 $\langle \text{bin} \rangle \quad * \quad \langle \text{exp} \rangle$
 $\downarrow \quad \swarrow \quad \downarrow \quad \searrow$
1 $\langle \text{factor} \rangle$ + $\langle \text{factor} \rangle$
 $\downarrow \quad \downarrow$
 $\langle \text{bin} \rangle \quad \langle \text{bin} \rangle$
 $\downarrow \quad \downarrow$
1 1 0

Use rules: $\langle \text{bin} \rangle ::= 1 \mid 0$

10/25/12

53

Example cont.

■ 1 * 1 + 0: $\langle \text{exp} \rangle$
 \downarrow
 $\langle \text{factor} \rangle$
 $\swarrow \quad \downarrow \quad \searrow$
 $\langle \text{bin} \rangle \quad * \quad \langle \text{exp} \rangle$
 $\downarrow \quad \swarrow \quad \downarrow \quad \searrow$
1 $\langle \text{factor} \rangle$ + $\langle \text{factor} \rangle$
 $\downarrow \quad \downarrow$
 $\langle \text{bin} \rangle \quad \langle \text{bin} \rangle$
 $\downarrow \quad \downarrow$
1 1 0

Fringe of tree is string generated by grammar

10/25/12

54

Your Turn: $1 * 0 + 0 * 1$

10/25/12

55

Parse Tree Data Structures

- Parse trees may be represented by OCaml datatypes
- One datatype for each nonterminal
- One constructor for each rule
- Defined as mutually recursive collection of datatype declarations

10/25/12

56

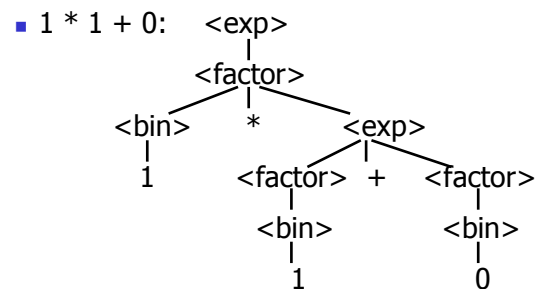
Example

- Recall grammar:
 $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle \mid \langle \text{factor} \rangle + \langle \text{factor} \rangle$
 $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle \mid \langle \text{bin} \rangle * \langle \text{exp} \rangle$
 $\langle \text{bin} \rangle ::= 0 \mid 1$
- type `exp` = `Factor2Exp` of `factor`
| `Plus` of `factor` * `factor`
and `factor` = `Bin2Factor` of `bin`
| `Mult` of `bin` * `exp`
and `bin` = `Zero` | `One`

10/25/12

57

Example cont.



10/25/12

58

Example cont.

- Can be represented as

```
Factor2Exp
(Mult(One,
      Plus(Bin2Factor One,
            Bin2Factor Zero))))
```

10/25/12

59

Ambiguous Grammars and Languages

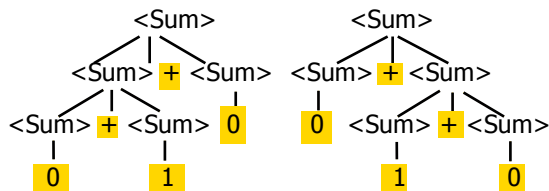
- A BNF grammar is *ambiguous* if its language contains strings for which there is more than one parse tree
- If all BNF's for a language are ambiguous then the language is *inherently ambiguous*

10/25/12

60

Example: Ambiguous Grammar

- 0 + 1 + 0



10/25/12

61

Example

- What is the result for:
3 + 4 * 5 + 6

10/25/12

62

Example

- What is the result for:
3 + 4 * 5 + 6
- Possible answers:
 - 41 = ((3 + 4) * 5) + 6
 - 47 = 3 + (4 * (5 + 6))
 - 29 = (3 + (4 * 5)) + 6 = 3 + ((4 * 5) + 6)
 - 77 = (3 + 4) * (5 + 6)

10/25/12

63

Example

- What is the value of:
7 - 5 - 2

10/25/12

64

Example

- What is the value of:
7 - 5 - 2
- Possible answers:
 - In Pascal, C++, SML assoc. left
7 - 5 - 2 = (7 - 5) - 2 = 0
 - In APL, associate to right
7 - 5 - 2 = 7 - (5 - 2) = 4

10/25/12

65

Two Major Sources of Ambiguity

- Lack of determination of operator precedence
- Lack of determination of operator associativity
- Not the only sources of ambiguity

10/25/12

66