

CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 3)



yourlastname.mail

- Please create a new file called `yourlastname.mail`
- It should contain one line:
`you@favorite.isp # First Last`
- New line (return) at end of line
- Copy to `~tiawatts/cs460drop`

```
chmod 644 yourlastname.mail
umask 033
cp yourlastname.mail ~tiawatts/cs460drop/
umask 077
```
- Thank you





Scheme

- Dialect of LISP
- MIT – mid-1970s
- Interpreted
- Expressions evaluated by the function EVAL
- Only primitives:
 - + (add), - (subtract), * (multiply), / (divide)
 - +, * may have zero or more operators
 - (+ 4 6) returns 10; (+) returns 0
 - (- 7 5) returns 0
 - (* 3 2 4) returns 24; (*) returns 1
 - (/ 6 8) returns $\frac{3}{4}$ or 0.75 (implementation dependent)



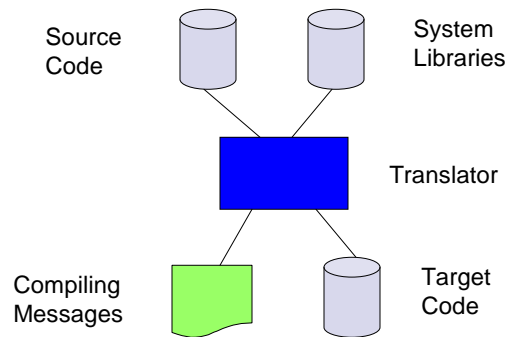
Language Development and Translation

- Describing the components of a language
 - Lexical components
 - Regular expressions and Finite Automata
 - Syntactical components
 - Grammars
 - Semantic Components
 - Attribute Grammars
 - Dynamic Semantics

Language Translation



System Libraries



Phases of Compilation



- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Intermediate code generation
- Optimization
- Target code generation

Phases of Compilation



- Front End Phases
 - Lexical analysis
 - Break stream of characters into lexemes
 - Generate “tokens”
 - Syntactical analysis
 - Determine and validate order of stream “tokens”
 - Semantic analysis
 - Determine meaning of stream of “tokens”
 - Intermediate code generation
 - Platform independent representation
- Language Dependent

Phases of Compilation



- Back End Phases
 - Optimization
 - Code reordering
 - Register allocation
 - Instruction level Parallelization
 - Target Code Generation
 - Machine Architecture Dependent

Compilation vs Interpretation

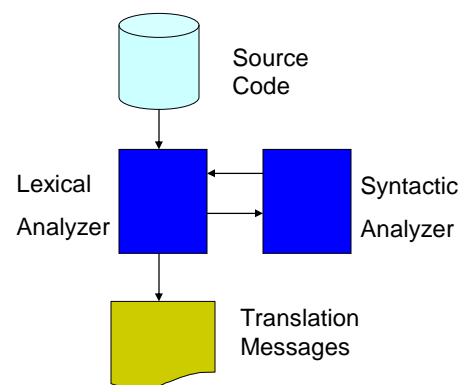


- **Compilation**
 - Complete translation of source program to target code
 - Run source code
 - Platform dependent
 - Production environment
- **Interpretation**
 - As needed translation of source program to target code
 - Interpreter needed to run
 - Platform independent
 - Debugging environment

Lexical Analyzer



- Recognize lexemes within character stream of source program
- Generates tokens
- Performs input and error output
- Under control of the syntactic analyzer





What is a “lexeme”?

- String of characters with a meaning
- Examples?
 - Key/Reserved words
 - User defined identifiers
 - Literals
 - Symbols – operators
- Defined using regular expressions
- Recognized by the implementation of a DFA



Regular Expressions

- Alphabet – the symbols that actually appear in the lexeme
- Special symbols to define the regular expression
 - () : grouping
 - * : 0 or more occurrences of a pattern
 - + : 1 or more occurrences of a pattern
 - | : indicates alternatives
 - λ : indicates nothing (lambda)

Regular Expression Examples



- Alphabet = {a,b,c}
- Examples
 - $a(b|c)a$
 - $a^+(b|c)a^+$
 - $a(b|c)^*a$
 - abc^*ba
 - $(a|b|c|\lambda)((ab^*c)|(cb^*a))^+$
- Regular expression for numeric literals?

Regular Expression for Numeric Literals



- Alphabet = ?
- Regular expression for integer?
- + or - ?
- Regular expression for real?
- + or - ?
- Regular expression for general class of numeric literals

Regular Expression for Numeric Literals



- Alphabet = $\{+, -, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Regular expression for integer
 - $(0|1|2|3|4|5|6|7|8|9)^+$
- + or - ?
 - $(+|-|\lambda)(0|1|2|3|4|5|6|7|8|9)^+$
- Regular expression for real?
 - $(0|1|2|3|4|5|6|7|8|9$
- + or - ?
 - $(+|-|\lambda)(0|1|2|3|4|5|6|7|8|9)^+ \cdot (0|1|2|3|4|5|6|7|8|9)^+$


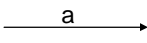
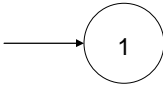

Regular Expression for Numeric Literals



- Regular expression for general class of numeric literals
 - $(+|-|\lambda)(0|1|2|3|4|5|6|7|8|9)^+ \cdot (0|1|2|3|4|5|6|7|8|9)^+|\lambda)$
- How do you recognize the end of a numeric literal?

Deterministic Finite Automata



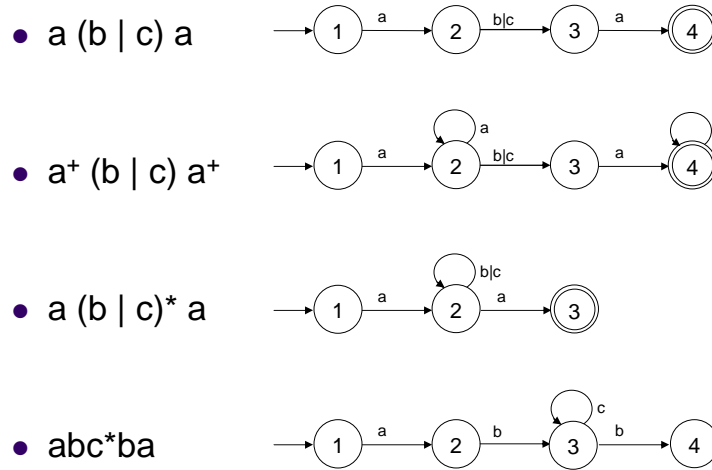
- States 
- Transitions 
- Start state 
- Final state 

DFAs for Examples



- $a(b | c)a$
- $a^+(b | c)a^+$
- $a(b | c)^*a$
- abc^*ba
- $(a|b|c|\lambda)((ab^*c)|(cb^*a))^+$

DFAs for Examples



$(a|b|c|\lambda)((ab^*c)|(cb^*a))^+ ?$

