CSE 4102

Programming Languages

$$(\lambda x.xx)(\lambda x.xx)$$

Overview

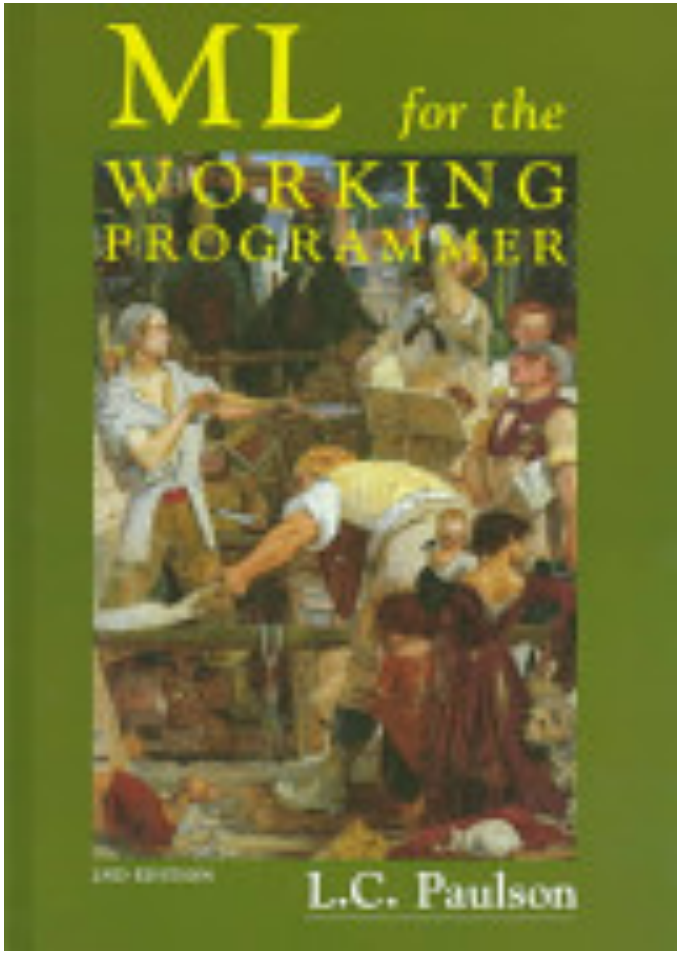
- Administrative details
- Course objective
  - Why study programming languages ?
  - Syllabus
- The Beginning

Administrative Details

- The material for this course is available online
  - http://courses.engr.uconn.edu/moodle
  - Simply register for the course (Key = CSE4102)
- The syllabus contains all the practical details
  - Office hours
  - Forum
  - Grading
  - Collaboration policy
- Class participation does matter
  - Use the moodle forum for discussion
  - All questions are welcome

Textbook

- ML For the working Programmer
  - Functional Programming
  - ML
  - Lambda calculus
- Does not cover the other topics
- **Not** mandatory **but** useful!



Course Objectives

- What are we trying to achieve ?
  - What is the role of programming languages ?
  - Exposure to several paradigms
  - Exposure to theory behind PL.
  - Exposure to pragmatics
- Is C++/Java/C# the end of the road ?
- Where are Programming Languages headed ?
  - New features
  - Resurgence of key abstractions
- Knowledge is power
  - With better tools, one writer better programs.

Syllabus

- Functional Programming
  - ML
  - Lambda calculus
- Procedural Paradigm
  - C

Interlude (midterm)

- Object-Oriented Programming
  - C++ | Smalltalk | Objective-C
- Logic & Constraint Programming

epilogue (final)

Starting Blocks...

- Today's Topics....
  - Why Bother ?
  - The Paradigm soup
  - Translation
  - Lexical structure and regular languages

The Arguments

- Some reasons
  - Expressive power
  - Better decision maker
  - Exposure & Dynamics
  - Language design for its own sake.

Expressiveness

- Purpose of languages
  - Bridge the gap between
    - Computer “language”
    - Human “language” / “thinking”
- Fact
  - The bigger the chasm...
  - The harder it is to express our ideas to the machine
- Solution
  - Bridge from both sides!

Expressiveness

- Programming Languages
  - Designed to solve “human-related” problems
  - Many similarities among languages
- Bottom line
  - Understand how/why a language was designed
  - To use it more effectively!
- Very often...
  - Desirable feature that is absent can be “emulated”

Decision Making

- Different problems require different tools.
  - Not everything is a nail....

> General purpose languages are not the solution since there is no such thing as a general purpose problem.
>
> OOPSLA'02 Anonymous.

- Being aware of the various tools help us choose the right one for the task at hand.
  - You *will* experience this during semester.

Exposure & Dynamics

- Computer science evolves fast
  - Some languages come and go in a couple of years...
  - Others stick around for decades....
- Change
  - Ability to evaluate new languages is useful to critically evaluate new comers.
  - Some features keep coming back. See the truth behind the smoke screen.

What is there is study ?

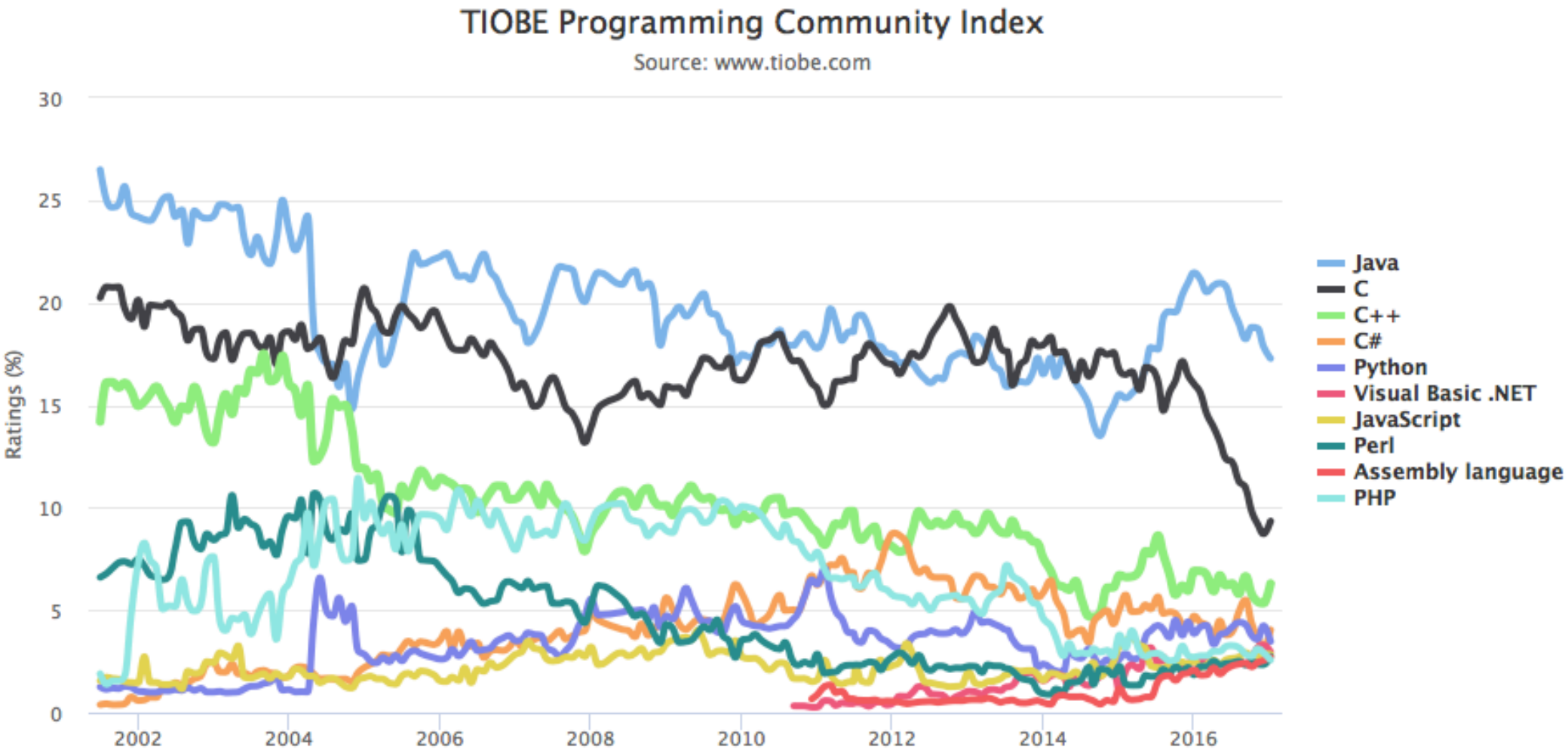
- Three aspects
  - Syntax
    - The form of a program, of its *sentences*
  - Semantics
    - The meaning of a program
  - Pragmatics
    - How to use programming languages features
    - What features make a difference.

Language Use

- Yearly popularity contest: TIOBE Index
  - (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

2017	Jan 2016	Change	Programming Language	Ratings	Change
1	1		Java	17.278%	-4.19%
2	2		C	9.349%	-6.69%
3	3		C++	6.301%	-0.61%
4	4		C#	4.039%	-0.67%
5	5		Python	3.465%	-0.39%
6	7	↑	Visual Basic .NET	2.960%	+0.38%
7	8	↑	JavaScript	2.850%	+0.29%
8	11	↑	Perl	2.750%	+0.91%
9	9		Assembly language	2.701%	+0.61%
10	6	↓↓	PHP	2.564%	-0.14%
11	12	↑	Delphi/Object Pascal	2.561%	+0.78%
12	10	↓	Ruby	2.546%	+0.50%
13	54	↑↑	Go	2.325%	+2.16%
14	14		Swift	1.932%	+0.57%
15	13	↓	Visual Basic	1.912%	+0.23%
16	19	↑	R	1.787%	+0.73%
17	26	↑↑	Dart	1.720%	+0.95%
18	18		Objective-C	1.617%	+0.54%
19	15	↓↓	MATLAB	1.578%	+0.35%
20	20		PL/SQL	1.539%	+0.52%

Long Term Trends



Learning Programming

- Trend
  - "Teach Yourself C++ in 24 hours"
  - Learn .... in .... days/hours
- Likely?
  - A nice read!
  - http://norvig.com/21-days.html

Norvig's comments

- So You Want to be a Programmer…
  - Get **interested** in programming,
    - and do some because it is fun.
  - **Program**. The best kind of learning is learning by doing.
  - **Talk** with other programmers; read other programs.
  - If you want, put in four years at a **college**
  - Work on **projects with** other programmers.
  - Work on **projects after** other programmers.
  - Learn at least a half dozen **programming languages**.
  - Remember that there is a "**computer**" in "computer science".
  - …

Learning Languages...

- Realistically
  - It takes years
  - Some even say decades....
- But
  - Lessons learned in a language
  - Often carry over to other language of the *same genre*

The Bare Bone

- First some historical context.
  - Von Neuman Architecture
    - Program & Data in a single memory
    - CPU fetches, decodes and execute instruction
    - CPU can have multiple functional units
    - A memory hierarchy can speed up resource access (caches)
  - This is all we need!

```
01000101110100101010100010101010101010010100100110100101010100010101010101101001.....
```

....Maybe not!

- Trend
  - Move away from machine language
  - Move closer to human language
- Motivation
  - "Language is limiting factor to what we can express and solve"
- First Move
  - Machine Language
    - Move to symbolic representation for
      - Instructions
      - Variables
    - Abstraction level
      - Essentially unchanged: machine level.

The First Big Step

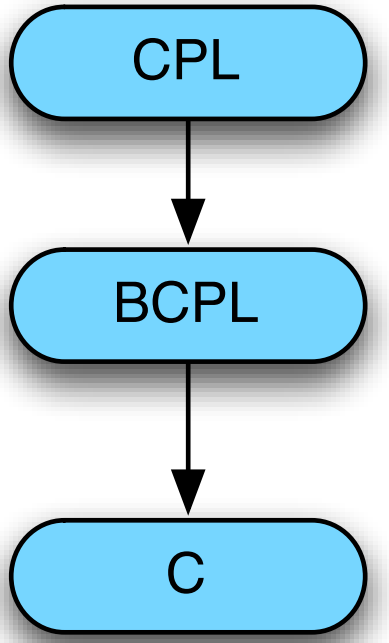
- Move to a higher level language
  - FORTRAN [1957~]
    - Readable familiar notation
    - Machine independent
    - Libraries
    - Program “verification”
  - An immediate strong success.
- On FORTRAN's heels
  - From Algol 60 to Lisp, Scheme,Prolog, C#, PHP, ASP,....
- Motto
  - Performance isn't everything...
  - Ease of use and expressiveness is more important.

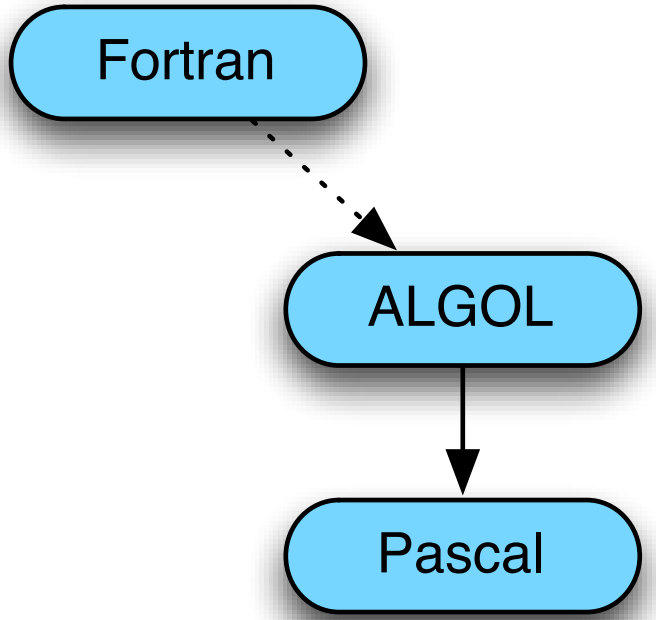
Paradigms [i.e., *genres*]

- Four major directions over the last 50 years
  - Procedural Oriented Programming [1955~]
  - Object Oriented Programming [1961~]
  - Functional Programming [1958~]
  - Logic Programming [1972~]

Procedural

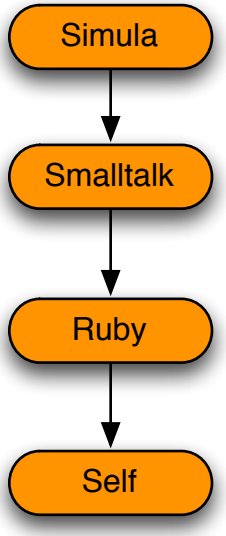
- Genealogy
- Motto
  - Computation Centric
  - Sequencing
  - Performance
  - Verification (Typed)
  - Store oriented





Object Oriented

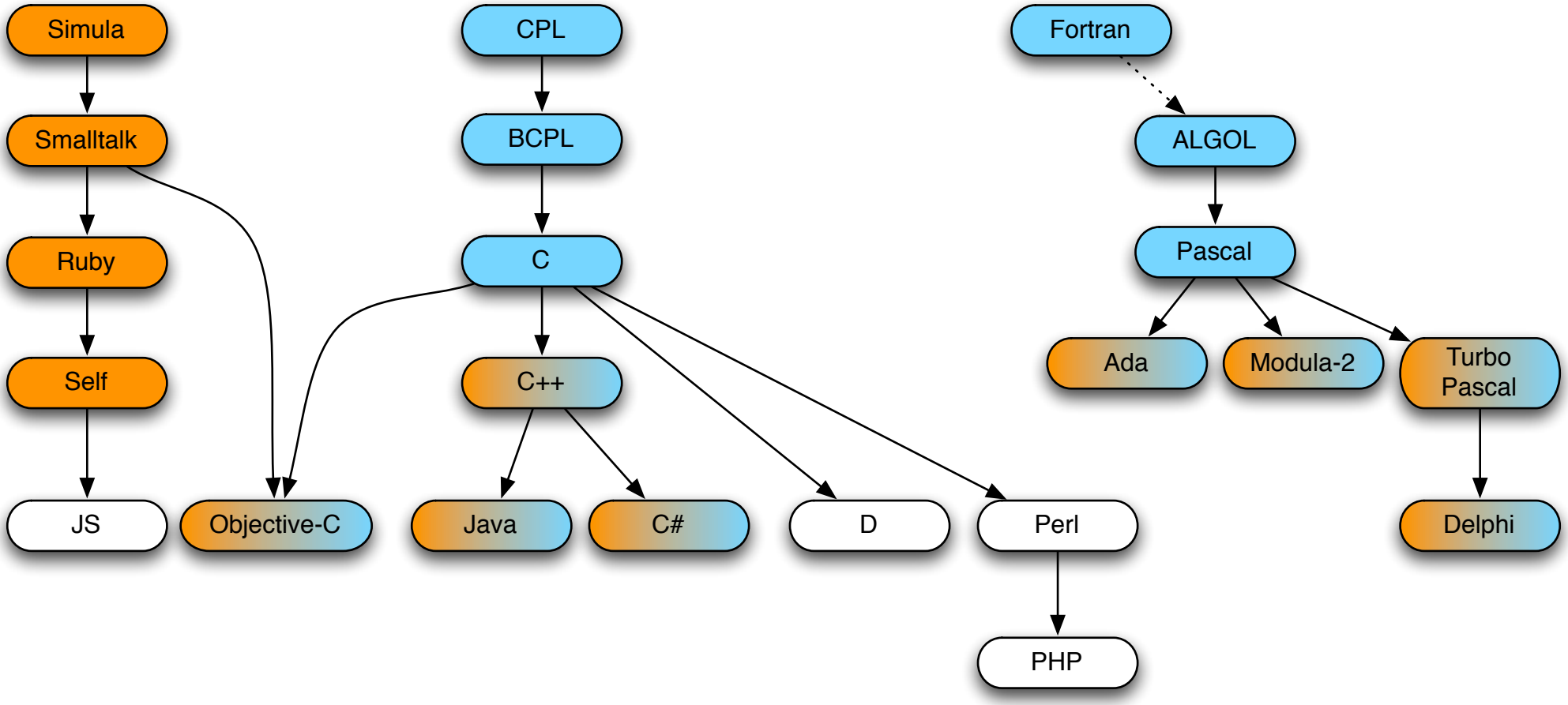
- Genealogy
  - K. Nygaard / O.J. Dahl
- Source: Simulation



- Motto
  - State Centric
  - Organize State with classes in hierarchies

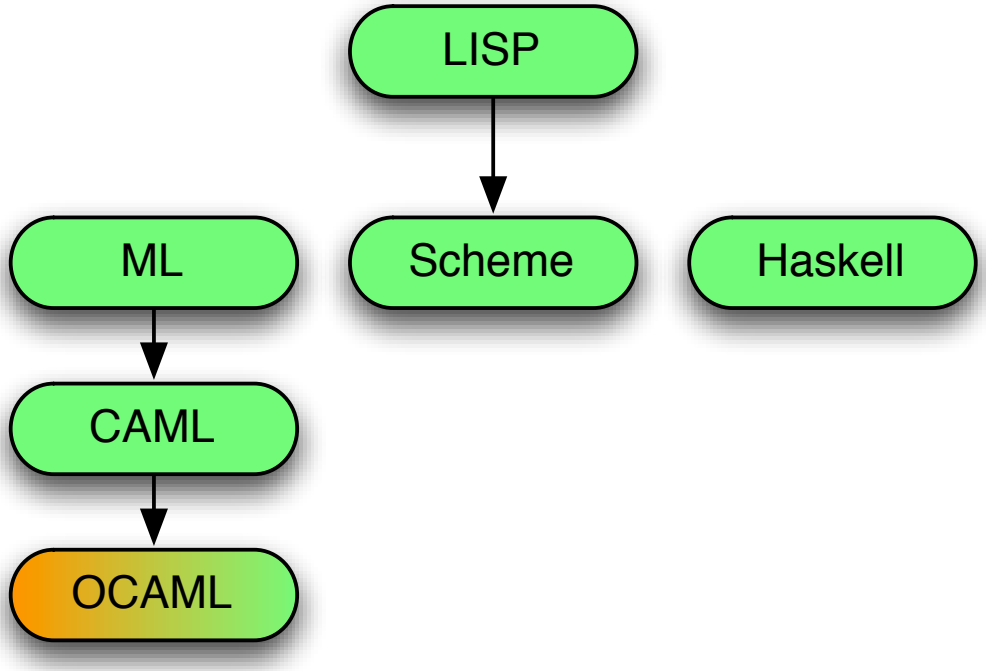
Object Oriented

- Genealogy
  - K. Nygaard / O.J. Dahl
- Source: Simulation
- Motto
  - State Centric
  - Organize State with classes in hierarchies



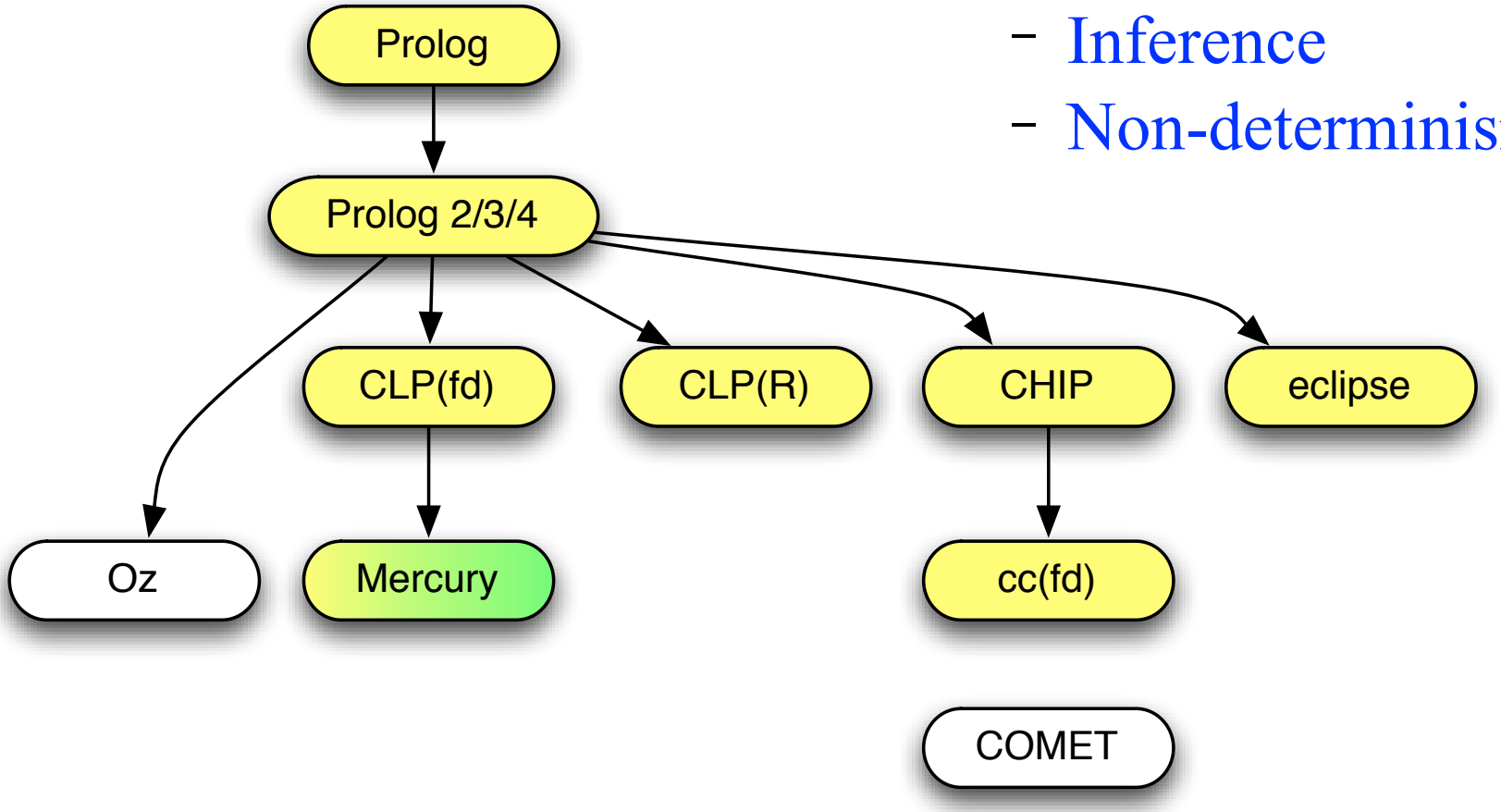
Functional Programming

- Genealogy
  - Mc Carthy
  - R. Milner
  - Haskell Curry
- Motto
  - Everything is a function
  - Symbolic computation
  - Assignments are **BAD**



Logic Programming

- Genealogy
  - Alain Colmerauer
- Motto
  - Computation is reasoning
  - First order & Predicate logic as premises
  - Inference
  - Non-determinism



Starting Blocks...

- Today's Topics....
  - Why Bother ?
  - The Paradigm soup
  - Translation
  - Lexical structure and regular languages

Layers...

SHREK:

For your information, there's a lot more to *computers* than people think.

DONKEY:

Example?

SHREK:

Example? Okay. Uh... *computers* are like onions.

DONKEY:

They stink?

SHREK:

Yes. No!

DONKEY:

Oh, they make you cry?

SHREK:

No!

DONKEY:

Oh, you leave them out in the sun, they get all brown and start sprouting little white hairs.

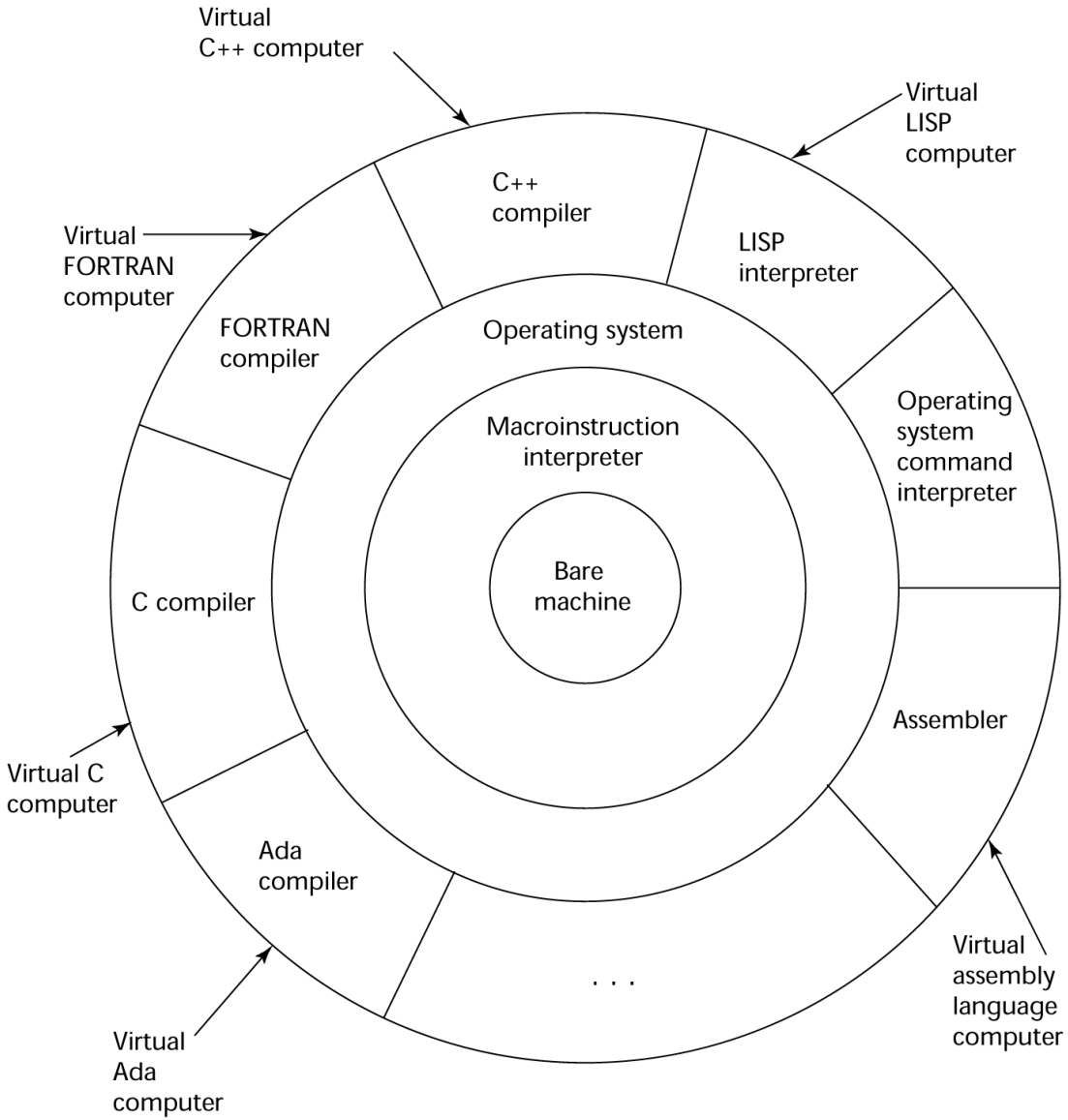
SHREK:

No! Layers! Onions have layers. *Computers* have layers. Onions have layers.

You get it? They both have layers.

Layers!

- Virtual Computers
- Issue
  - How to go
    - From upper layers
    - To deeper layers ?

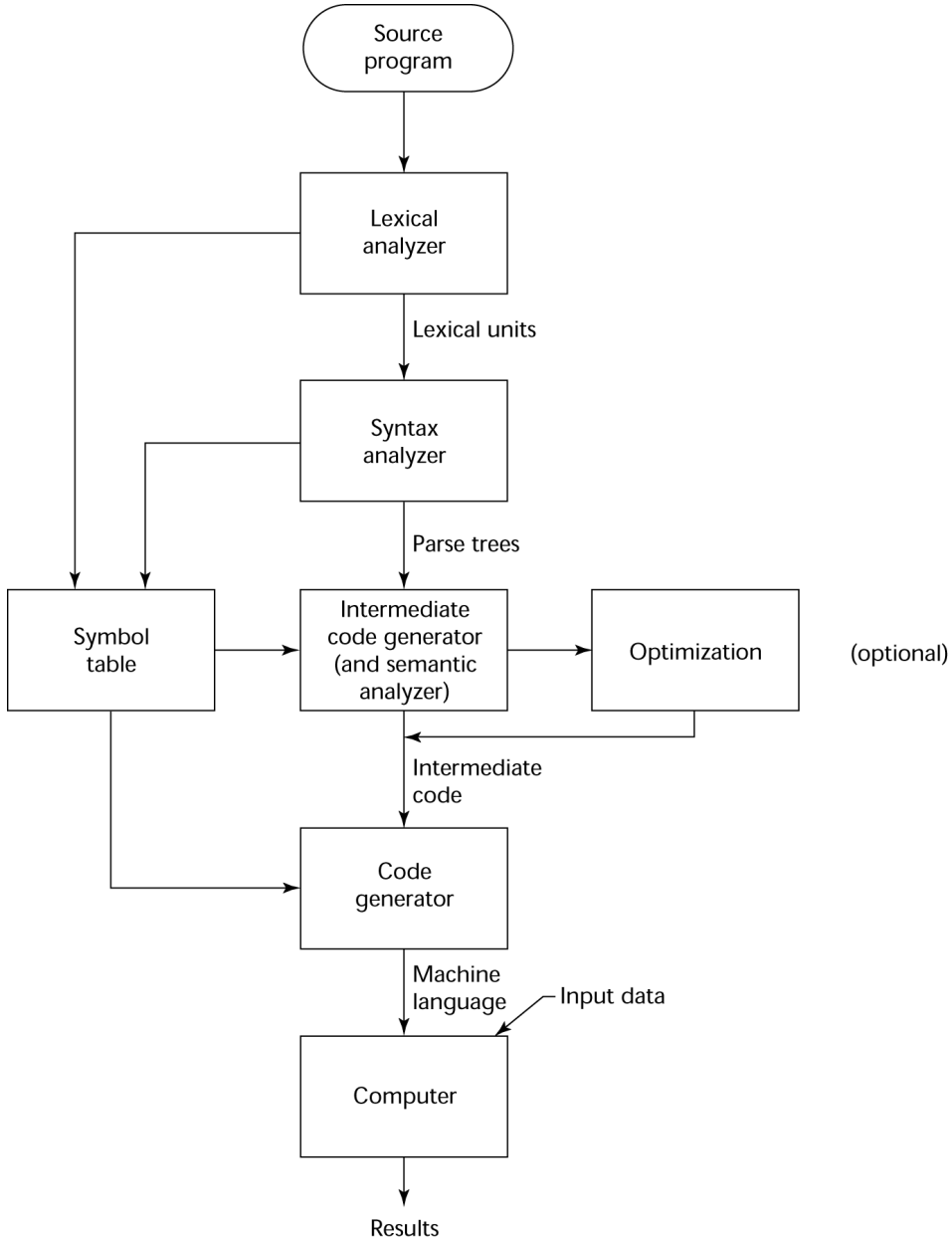


Translation

- Translation is required!
- Three possibilities
  - Interpreter
  - Compiler
  - Mixed

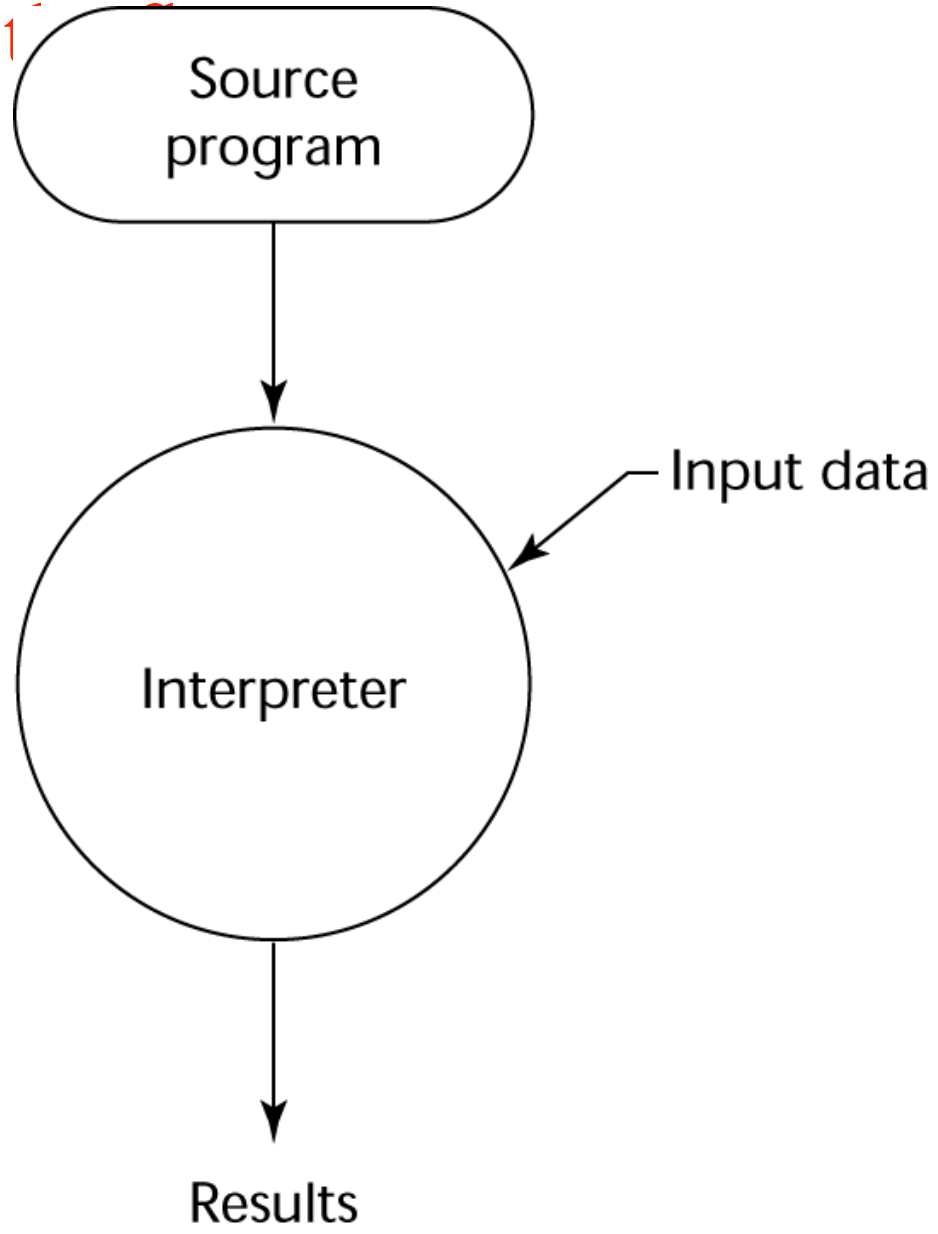
Compiler

- Several phases
  - Lexical analysis
  - Parsing
  - Semantic checking
  - Code Generation
    - Possible targets ?
      - Machine code
      - C
      - Any other language...



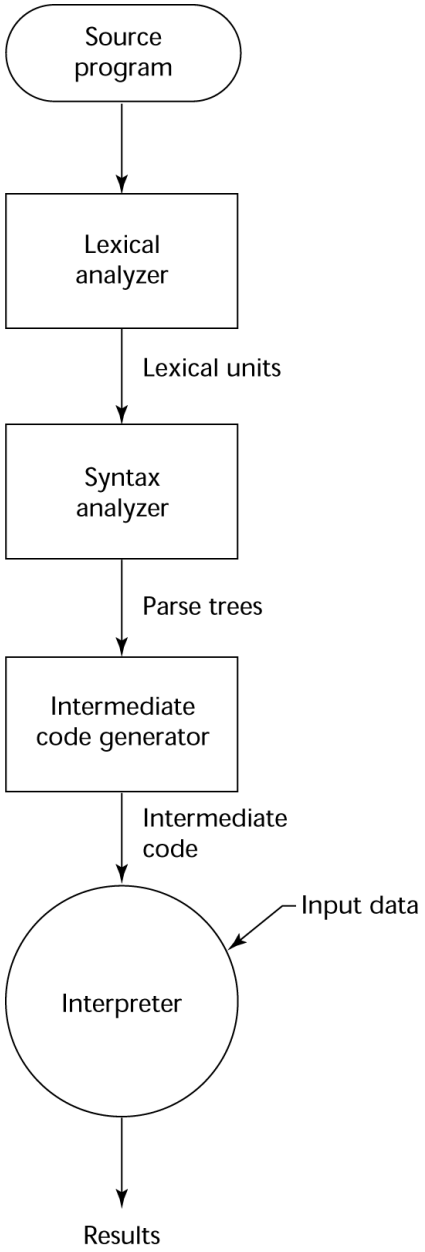
Interpreter

- Translate statements on t
- Several phases
  - Lexical analysis
  - Parsing
  - Semantic checking
  - Interpretation



Mixed

- Basic Idea
  - At compile time
    - Do the scanning/parsing/analysis once
    - Generate Intermediate Representation
  - At runtime
    - Read the IR
    - Interpret the IR
  - Example ?



Starting Blocks...

- Today's Topics....
  - Why Bother ?
  - The Paradigm soup
  - Translation
  - Lexical structure and regular languages

Lexical Analysis

- Objective
  - Turn a stream of symbols into....
  - A stream of Tokens
- Rationale
  - Details of token representation are irrelevant
- How to do it ?
  - Tokens are strings of symbols
    - Belong to a specific language
    - Language is Regular
    - Use Finite State Automata

Regular Expression

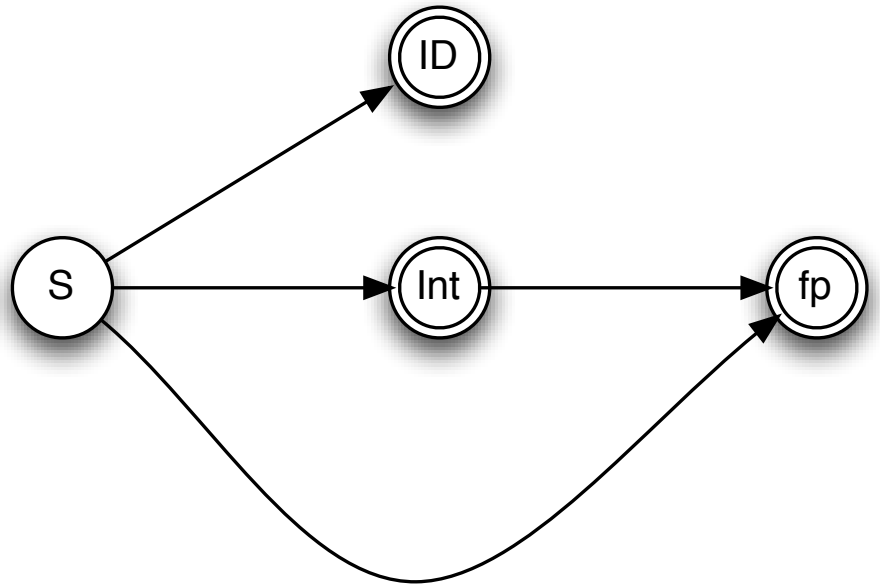
- Alphabet

  $$\Sigma = \{a, b, c, \cdots, 0, 1, 2, \cdots, 9, ?, \cdots\}$$

- Inductive definition
  - Symbols — $\forall x \in \Sigma : x$ is a R.E.
  - Empty string — "" is a R.E.
  - Concatenation — $\forall r, l \in$ R.E. rl is a R.E.
  - Alternatives — $\forall r, l \in$ R.E. $r|l$ is a R.E.
  - Closure — $\forall r \in$ R.E. r^* is a R.E.
- Examples
  - Identifier — [a-zA-Z][a-zA-Z0-9_]*
  - Integer — [0-9]+
  - Fixed Floating Point — [0-9]*.[0-9]+

Finite State Automata

- Is a compiled Regular Expression!
- Example
  - Integer & ID & Fixed Notation FP
  - Compose the 3 R.E. As
    - ([a-zA-Z][a-zA-Z0-9_]* | [0-9]+ | [0-9]*.[0-9]+)
  - Turn it into an FSA



Parsing

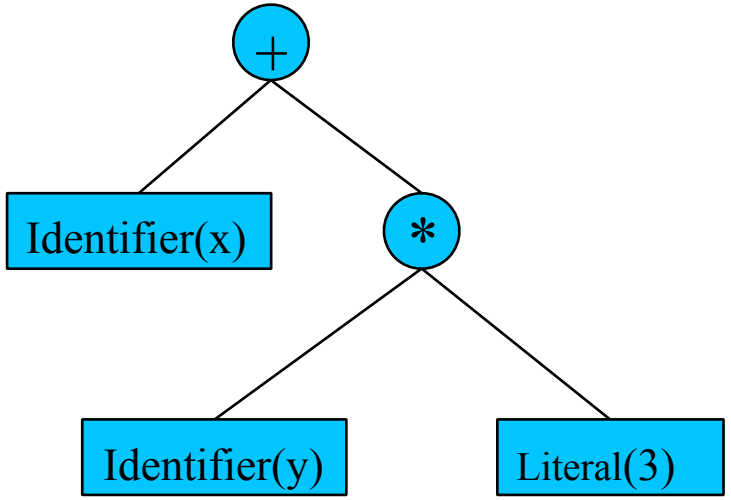
- Sequence of tokens form sentences
  - Sentences have a structure
  - Structure is very regular and obeys grammatical rules
  - Structure is tree-oriented

Example

- What we write
  - Expression: x+y*3
- The token stream we see
  - Identifier(“x”) + Identifier(“y”) * Literal(3)
- The grammatical rules we apply

```
Expr    ::= Expr + Expr
        ::= Expr * Expr
        ::= Term
Term    ::= Identifier
        ::= Literal
```

- The phrase we mean



Grammars

- All languages have grammars
  - BNF with
    - Associativity rules
    - Precedence rules
  - Grammars for formal languages are well behaved
    - Context Free
- I will write
  - Strings of characters
- But I will think about
  - Trees!

From Token Stream to Trees

- Finite State Automata are
  - A machine to interpret regular expression
- Is there a machine to interpret context-free grammars ?
  - Good News:
    - Yes!
  - Pushdown automata
    - A stack-based machine that takes as input
      - A token stream
      - A grammar
    - And produces
      - An abstract syntax tree.

Connection

- What to know more on translation ?
  - Take a compiler class!
    - Learn everything about practical aspects of R.E.
    - Learn about all the techniques
      - for defining grammars
      - Parsing
    - Learn about semantic analysis
    - Learn about code generation
- What we are concerned about in this class
  - What languages have to offer
    - Semantics [meaning]
  - We cannot study languages without
    - Understanding their grammar, i.e., the structure of the phrases one can write.

Next Time

- Topics for next lecture
  - Functional Paradigm