Unnatural Language Processing
Natural Languages

A *natural language* is a form of communication peculiar to humankind. [Wikipedia]

Popular spoken natural languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese</td>
<td>1,205m</td>
</tr>
<tr>
<td>Spanish</td>
<td>322m</td>
</tr>
<tr>
<td>English</td>
<td>309m</td>
</tr>
<tr>
<td>Arabic</td>
<td>206m</td>
</tr>
<tr>
<td>Hindi</td>
<td>108m</td>
</tr>
<tr>
<td>Portuguese</td>
<td>178m</td>
</tr>
<tr>
<td>Bengali</td>
<td>171m</td>
</tr>
<tr>
<td>Russian</td>
<td>145m</td>
</tr>
<tr>
<td>Japanese</td>
<td>122m</td>
</tr>
<tr>
<td>German</td>
<td>95m</td>
</tr>
</tbody>
</table>

[Wikipedia]

Ethnologue catalogs 6,912 known living languages.
Conlangs: Made-Up Languages

Okrent lists 500 invented languages including:

- **Lingua Ignota** [Hildegard of Bingen, c. 1150]
- **Esperanto** [L. Zamenhof, 1887]
- **Klingon** [M. Okrand, 1984]
  
  Huq Us'pty G'm (I love you)

- **Proto-Central Mountain** [J. Burke, 2007]

- **Dritok** [D. Boozer, 2007]
  
  Language of the Drushek, long-tailed beings with large ears and no vocal cords

[Arika Okrent, *In the Land of Invented Languages*, 2009]
[http://www.inthelandofinventedlanguages.com]
Programming Languages

**Programming languages** are notations for describing computations to people and to machines.

Underlying every programming language is a **model of computation:**

- **Procedural:** C, C++, C#, Java
- **Declarative:** SQL
- **Logic:** Prolog
- **Functional:** Haskell
- **Scripting:** AWK, Perl, Python, Ruby
Programming Languages

There are many thousands of programming languages.

Tiobe’s ten most popular languages for May 2009:

1. Java
2. C
3. C++
4. PHP
5. Visual Basic
6. Python
7. C#
8. JavaScript
9. Perl
10. Ruby

[http://www.tiobe.com]

http://www.99-bottles-of-beer.net has programs in 1,271 different programming languages to print out the lyrics to “99 Bottles of Beer.”
"99 Bottles of Beer"

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.

[Traditional]
“99 Bottles of Beer” in AWK

BEGIN {
    for(i = 99; i >= 0; i--) {
        print ubottle(i), "on the wall," , lbottle(i) "."
        print action(i), lbottle(inext(i)), "on the wall."
    }
}

function ubottle(n) {
    return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s" : ")
}

function lbottle(n) {
    return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s" : "")
}

function action(n) {
    return printf("%s", n ? "Take one down and pass it around," : "Go to the store and buy some more,"")
}

function inext(n) {
    return n ? n - 1 : 99
}

“99 Bottles of Beer” in Perl

[Andrew Savage, http://search.cpan.org/dist/Acme-EyeDrops/lib/Acme/EyeDrops.pm]
“99 Bottles of Beer” in the Whitespace Language

[Edwin Brady and Chris Morris, U. Durham]
A Little Bit of Formal Language Theory

An *alphabet* is a finite set of symbols.

\{0, 1\}, ASCII, UNICODE

A *string* is a finite sequence of symbols.

\(\varepsilon\) (the empty string), 0101, dog, cat

A *language* is a countably infinite set of strings called sentences.

\[ \{ a^n b^n \mid n \geq 0 \}, \{ s \mid s \text{ is a Java program} \}, \{ s \mid s \text{ is an English sentence} \} \]

A language has properties such as a *syntax* and *semantics*. 
Language Translation

Given a source language $S$, a target language $T$, and a sentence $s$ in $S$, map $s$ into a sentence $t$ in $T$ that has the same meaning as $s$. 
Specifying Syntax: Regular Sets

Regular expressions generate the regular sets

\[ a(a|b)^* \] generates all strings of \( a \)'s and \( b \)'s beginning with an \( a \)

Finite automata recognize the regular sets

![Finiteautomaton.png](attachment:Finiteautomaton.png)
Some Regular Sets

All words with the vowels in order

facetiously

All words with the letters in increasing lexicographic order

aegilops

All words with no letter occurring more than once

dermatoglyphics

Comments in the programming language C

/* any string without a star followed by a slash */
Some Regular Expression Pattern-Matching Tools

egrep

egrep 'a.*e.*i.*o.*u.*y' /usr/dict/words

AWK
C
Java
JavaScript
Lex
Perl
Python
Ruby
Context-Free Languages

**Context-free grammars** generate the CFLs

Let $G$ be the grammar with productions $S \rightarrow aSbS | bSaS | \varepsilon$.

The language denoted by $G$ is all strings of $a$’s and $b$’s with the same number of $a$’s as $b$’s.

**Parsing algorithms** for recognizing the CFLs

- Earley’s algorithm
- Cocke-Younger-Kasami algorithm
- Top-down LL(k) parsers
- Bottom-up LR(k) parsers
Ambiguity in Grammars

Grammar \( S \rightarrow aSbS \mid bSaS \mid \epsilon \) generates all strings of \( a \)'s and \( b \)'s with the same number of \( a \)'s as \( b \)'s.

This grammar is ambiguous: \( abab \) has two parse trees.

\[
(ab)^n \text{ has } \frac{1}{n+1} \binom{2n}{n} \text{ parse trees}
\]
Programming Languages are not Inherently Ambiguous

The grammar \( G \) generates the same language

\[
S \rightarrow aAbS \mid bBaS \mid \varepsilon \\
A \rightarrow aAbA \mid \varepsilon \\
B \rightarrow bBaB \mid \varepsilon
\]

\( G \) is unambiguous and has only one parse tree for every sentence in \( L(G) \).
Natural Languages are Inherently Ambiguous

*I made her duck.*

[5 meanings: D. Jurafsky and J. Martin, 2000]

*One morning I shot an elephant in my pajamas. How he got into my pajamas I don’t know.*


*List the sales of the products produced in 1973 with the products produced in 1972.*

[455 parses: W. Martin, K. Church, R. Patil, 1987]
Methods for Specifying the Semantics of Programming Languages

Operational semantics

translation of program constructs to an understood language

Axiomatic semantics

assertions called preconditions and postconditions specify the properties of statements

Denotational semantics

semantic functions map syntactic objects to semantic values
Translation of Programming Languages

source program → Compiler → target program

input → Compiler → output

Compilers
Principles, Techniques, & Tools
Second Edition
Target Languages

Another programming language
CISCs
RISCs
Vector machines
Multicores
GPUs
Quantum computers
An Interpreter Directly Executes a Source Program on its Input
Java Compiler

source program

Translator

intermediate representation

input

Java Virtual Machine

output
Phases of a Classical Compiler

source program


token stream → syntax tree → annotated syntax tree → interim. rep. → interim. rep.

Symbol Table
Compiler Component Generators

lex specification

Lexical Analyzer Generator (lex)

source program

Lexical Analyzer

token stream

Syntax Analyzer

Syntax Analyzer Generator (yacc)

syntax tree
Lex Specification for a Desk Calculator

number: \([0-9]+\.?|[0-9]*\.[0-9]+\)

\%

[ ] { /* skip blanks */ }  

{number} { sscanf(yytext, "%lf", &yylval);  
  return NUMBER; }  

\n|. { return yytext[0]; }
Yacc Specification for a Desk Calculator

%token NUMBER
%left '+'
%left '*'

lines : lines expr '\n' { printf("%g\n", $2); }
| /* empty */
;

expr : expr '+' expr { $$ = $1 + $3; }
| expr '*' expr { $$ = $1 * $3; }
| '(' expr ')' { $$ = $2; }
| NUMBER
;

#include "lex.yy.c"
Creating the Desk Calculator

Invoke the commands

```bash
lex desk.l
yacc desk.y
cc y.tab.c -ly -ll
```

Result

```
1.2 * (3.4 + 5.6) → 10.8
```

Desk Calculator