Compilers

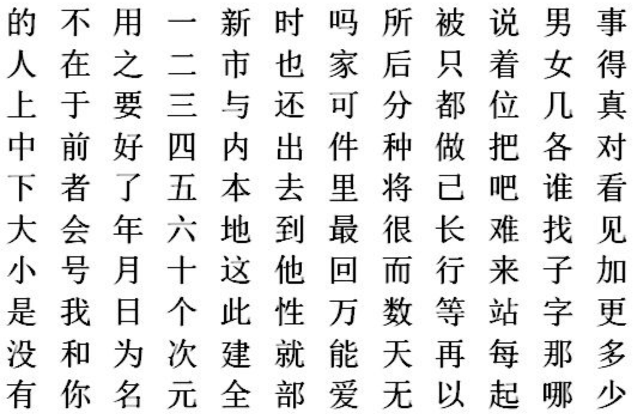
CSCI 4020U

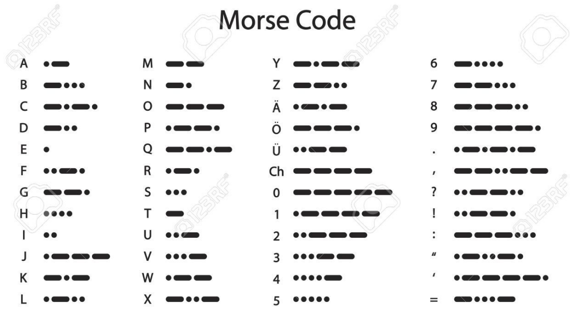
Strings and Languages

This course is about communication. In particular, we are interested in the *communication with machines* and *communication between machines*.

Symbols and Strings

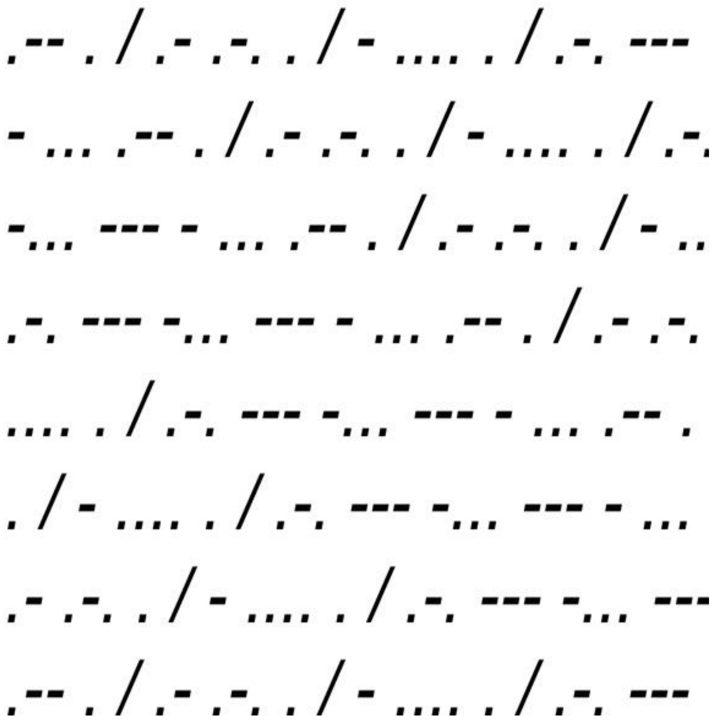
- An *alphabet* is a collection of *symbols.*
- Some alphabets are small
    a. Morse code has two symbols
    b. The english alphabet has 26 symbols.
- Some alphabets are quite large.
    a. Literate Chinese alphabet has over 4000 symbols.





Symbols and Strings

- No matter how large an alphabet is, it must be finite.
- We cannot encode *all* meaningful messages using *single* symbols because there are at least *infinitely many* meaning messages.
- We need strings.



Some Definitions

An alphabet Σ is an *finite set* of symbols.

A string s is a *finite sequence* of symbols from some alphabet.

A language L is a set of strings. Most useful languages are infinite sets.

The set of all possible strings from an alphabet Σ is denoted $\Sigma^\star$.

Fact

$$L \text{ is a language} \Rightarrow L \subseteq \Sigma^*$$

More Definitions

A string with zero symbols is written ε (epsilon).

A string is *valid* with respect to a language L if $s \in L$. Otherwise, we say that s is not valid w.r.t. L.

Example: English vs French

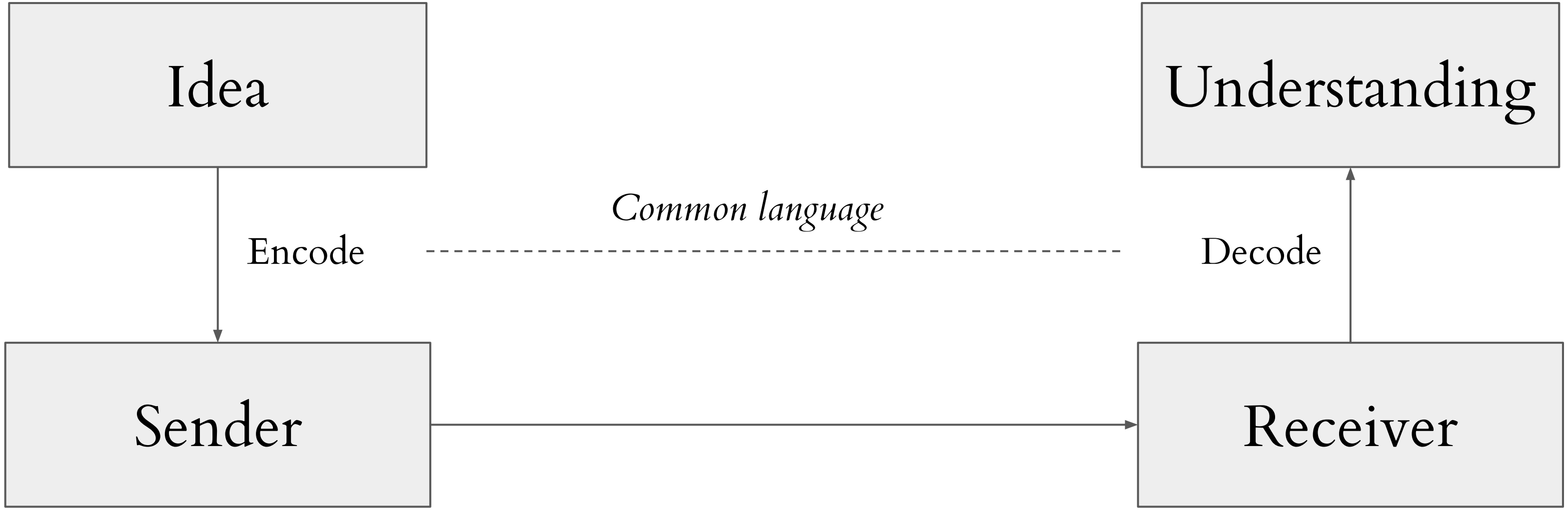
Σ_{EN} = {a, b, c, ... A, B, C, ... Z } ∪ Puntunations

Σ_{FR}={a,à,b,c,d,e,è,é,f,...z,A,À,B,C,D,E,...Z}
∪ Puntunations

Bonjour à tous ∈ French, so it's a valid French string

Bonjour à tous ∉ English, so it's not a valid English string.

Communication



Challenges

Bootstrapping communication

- How do we have the sender and receiver agree on using a common language?
- What is the language used to communicate the language to be used?

Expressiveness of language

- What type of messages are permitted by the common language used between the sender and the receiver?
- Are there *ideas* that cannot be communicated by the language?

Computation

Turing Machine

- Tape contains strings of symbols.
- Control logic is generating strings of a specific language.
- The control logic itself is defined as a string in the language understood by the Universal Turing Machine.

Lambda Calculus

- Expressions are strings of a language.
- String rewriting rules generate more strings in the language.

Programming

Sender = Programmer

- **Idea** is an algorithm to solve some problem.
- **Message** is the source code of a particular *language* that implements the algorithm.

Receiver = Computer

- **The decoder** turns the message (source code) into computational instructions (also of some language)
- **Understanding** is the actual computation triggered by the programmer.

Some languages we care about

Python Language

```
    year_names= []
    with open(filename, 'r') as baby_file:
        lines = baby_file.readlines()
        for line in lines:
            if '<h3 align="center">Popularity' in line:
                year = re.search('(\d{4})', line)
                print(year.group(0))
                continue

            rank_info = re.search('<td>(\d+)</td><td>(\w+)</td><td>(\w+)</td>', line)
            if rank_info is not None:
                print(rank_info.group(0))
                rank, boy, girl = rank_info.group(1), rank_info.group(2), rank_info.group(3)
                year_names.extend([boy + ' ' + rank,
                                   girl + ' ' + rank])

    extract_names = [year] + sorted(year_names)
    print(extracted_names)


extract_names('baby1990.html')
```

Some languages we care about

Clojure

```
(extend-type AdjacencyList
  Graph
  (out-degree [vtx graph]
    (count (graph vtx)))
  (in-degree [vtx graph]
    (count (for [v graph :when (some #{(:end vtx)} v)] v)))
  (bfs [G vtx Q visitf vals]
    ;;; Recursive implemtation of breadth-first search
    (letfn [(bfs [G vtx Q visitf vals]
              (let [adj ((:al G) vtx)]
                (if (and (empty? adj)
                         (empty? Q))
                  (conj vals (visitf (:val vtx)))
                  (let [Q (into Q adj)]
                    (recur G (peek Q) (pop Q) visitf (conj vals (visitf (:val vtx))))))))]
      (bfs G vtx Q visitf vals))))
```

Some languages we care about

C

```c
/* C P  C H E C K  E N D  R I G H T */
/* %%Function:CpCheckEndRight %%Owner:chic */
CP CpCheckEndRight(cp, cpAnchor, pflss, psel, psty, fExtend)
CP cp;
CP cpAnchor;
struct FLSS *pflss;
struct SEL *psel;
int *psty;
BOOL fExtend;
{

    /* check for special case: insert point will not be placed to the right of
        an end of paragraph */
    int chBreak = pflss->chBreak;
    if (cp == pflss->cpMac && (chBreak == chEop ||
            ((*hwwdCur)->fPageView && chBreak == chSect) || chBreak == chTable
            || chBreak == chCRJ) &&
            (psel->fSelAtPara || (*psty <= stySent &&
            !fExtend || cp < cpAnchor/* backward extension */)))
```

Some languages we care about

JVM Bytecode

```
 0 aload_0
 1 new #3 <acceptanceTests/treeset_personOK/Main$A>
 4 dup
 5 new #8 <java/lang/Object>
 8 dup
 9 invokespecial #10 <java/lang/Object.<init>>
12 new #12 <java/lang/Integer>
15 dup
16 iconst_2
17 invokespecial #14 <java/lang/Integer.<init>>
20 invokespecial #17 <acceptanceTests/treeset_personOK/Main$A.<init>>
23 new #12 <java/lang/Integer>
26 dup
27 iconst_1
28 invokespecial #14 <java/lang/Integer.<init>>
31 invokespecial #17 <acceptanceTests/treeset_personOK/Main$A.<init>>
34 getstatic #20 <java/lang/System.out>
37 new #3 <acceptanceTests/treeset_personOK/Main$A>
40 dup
41 new #8 <java/lang/Object>
44 dup
45 invokespecial #10 <java/lang/Object.<init>>
48 new #12 <java/lang/Integer>
51 dup
52 iconst_2
53 invokespecial #14 <java/lang/Integer.<init>>
56 invokespecial #17 <acceptanceTests/treeset_personOK/Main$A.<init>>
59 invokevirtual #26 <java/io/PrintStream.println>
62 return
```

Some languages we care about

OP Code for CPU

```
        _ATLTRY
        {
1052D206 C7 45 FC 00 00 00 00 mov          dword ptr [ebp-4],0
            if(IsSingleThreadedApartment())
1052D20D E8 8C 38 B4 FF       call         IsSingleThreadedApartment (10070A9Eh)
1052D212 85 C0                test         eax,eax
1052D214 0F 84 BF 00 00 00    je           CPlaybackEx::FinalConstruct+119h (1052D2D9h)
                __E(m_EventWindow.Create());
1052D21A 6A 00                push         0
1052D21C 51                   push         ecx
1052D21D 8B CC                mov          ecx,esp
1052D21F 89 A5 AC FE FF FF    mov          dword ptr [ebp-154h],esp
1052D225 6A 00                push         0
1052D227 E8 DB ED B3 FF       call         ATL::_U_MENUorID::_U_MENUorID (1006C007h)
1052D22C 89 85 A4 FE FF FF    mov          dword ptr [ebp-15Ch],eax
1052D232 6A 00                push         0
1052D234 6A 00                push         0
1052D236 6A 00                push         0
1052D238 51                   push         ecx
1052D239 8B CC                mov          ecx,esp
1052D23B 89 A5 B8 FE FF FF    mov          dword ptr [ebp-148h],esp
1052D241 6A 00                push         0
```

The journey of source code

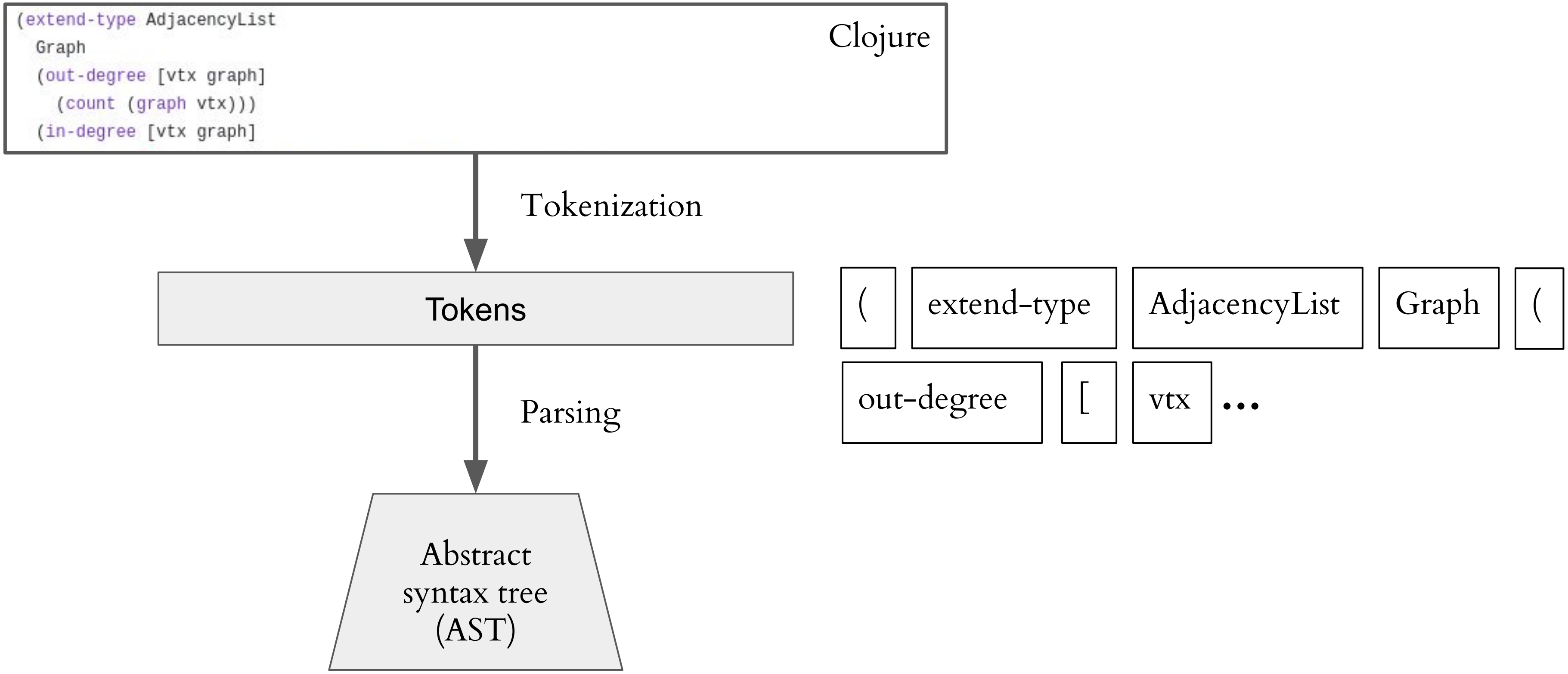
Graph adjacency list

↓

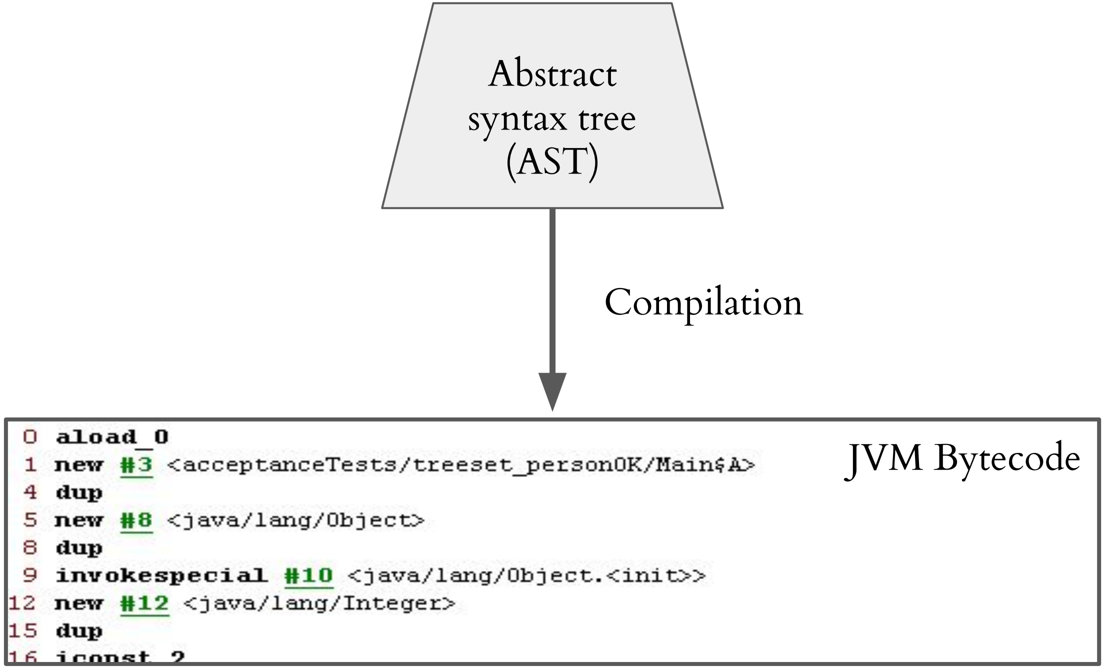
Clojure

```clojure
(extend-type AdjacencyList
  Graph
  (out-degree [vtx graph]
    (count (graph vtx)))
  (in-degree [vtx graph]
    (count (for [v graph :when (some #{(:end vtx)} v)] v)))
  (bfs [G vtx Q visitf vals]
    ;;; Recursive implemtation of breadth-first search
    (letfn [(bfs [G vtx Q visitf vals]
              (let [adj ((:al G) vtx)]
                (if (and (empty? adj)
                         (empty? Q))
                  (conj vals (visitf (:val vtx)))
                  (let [Q (into Q adj)]
                    (recur G (peek Q) (pop Q) visitf (conj vals (visitf (:val vtx))))))))]
      (bfs G vtx Q visitf vals))))
```

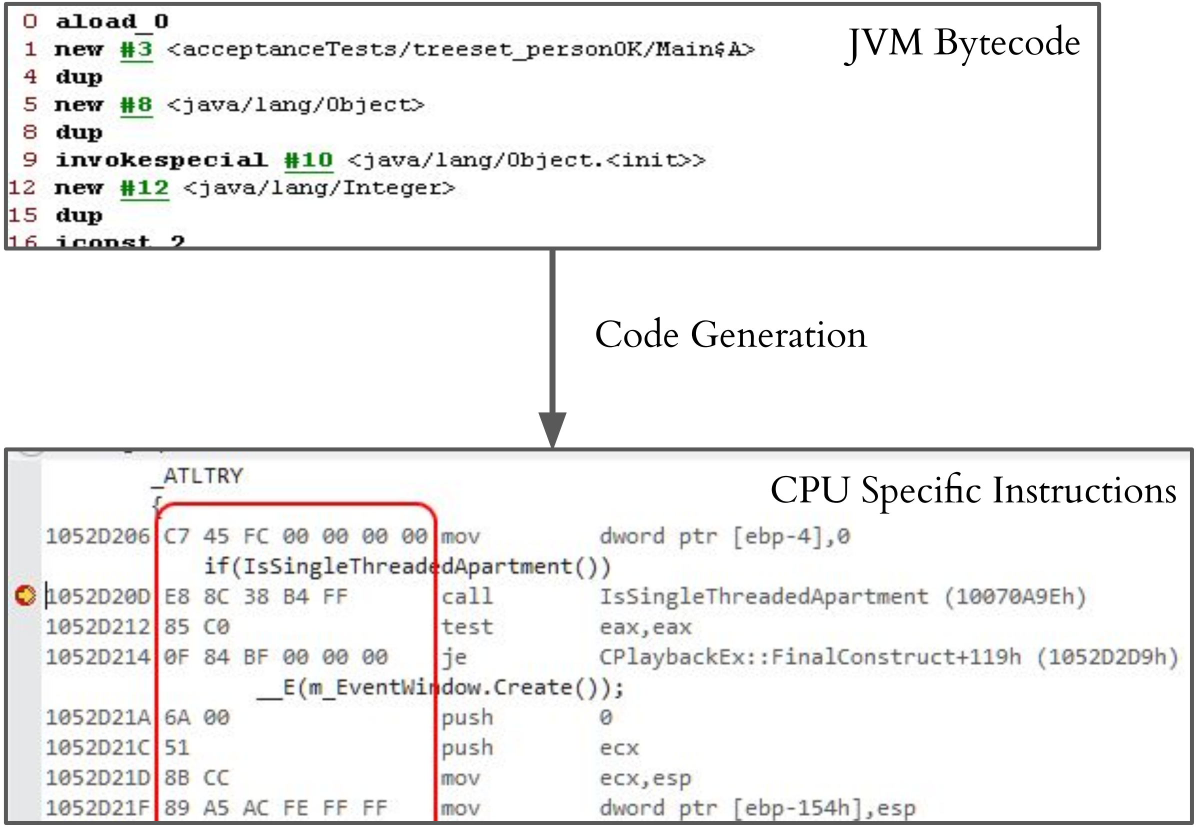
The journey of source code



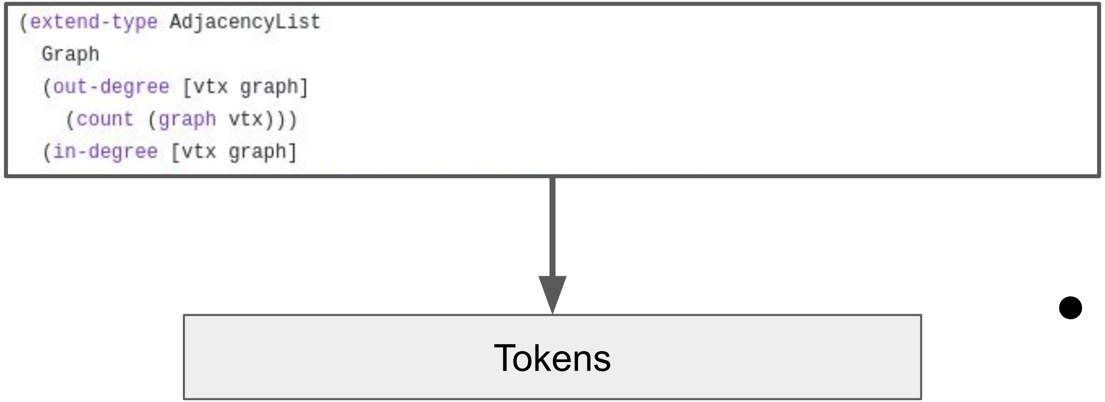
The journey of source code



The journey of source code



Languages everywhere



- How many types of tokens should we have?
- What are the tokens?

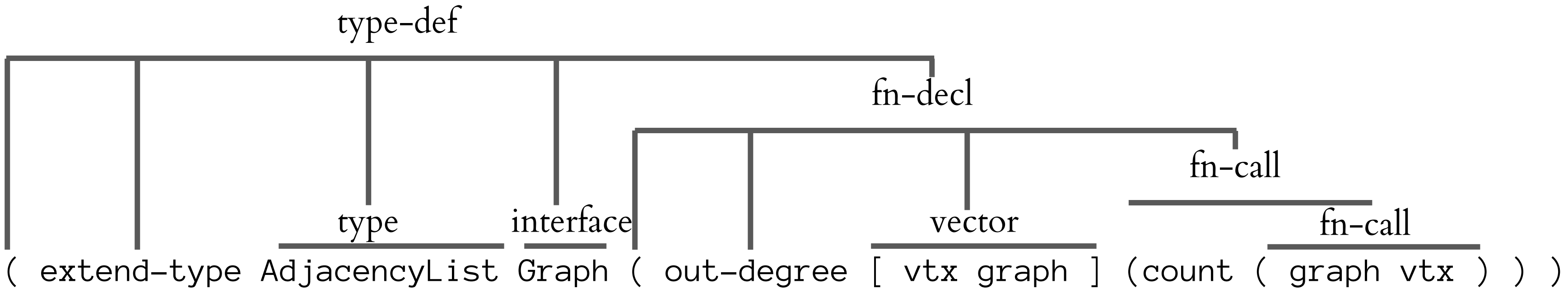
Regular Languages

Languages everywhere

```
(extend-type AdjacencyList
  Graph
  (out-degree [vtx graph]
    (count (graph vtx)))
  (in-degree [vtx graph]
```

- What are the valid sequences of tokens?
- How should we organize tokens into semantic groups?

Context Free Languages



About this course

Languages and Parsing

1. Regular Languages
2. Regular Expression and Automata
3. Context Free Languages
4. Parse Trees
5. Parsing Algorithms

Programming

1. Computation by programming in interpreted languages
2. Interpreter construction
3. Computation by programming in compiled languages
4. Three address bytecode
5. Compiler construction

About this course

1. We use Antlr, a parser generator library for Java.
2. We program in Java.
3. We program in Kotlin.
4. We provide cloud-based computing environment.