Programming language

* Human and machine-readable notational system primarily intended to facilitate human-machine interaction
* Like all languages, has a *syntax*, and elements have *semantics*
* Produces program
  + Structured entity with semantics

Formal approaches:

* A formal and quantitative characterization of a programming language is based upon a formal language which itself is based upon a formal grammar
  + Denoted *G*
  + This viewpoint allows us to alternative and and quantitatively define a program as a *sentence* or *string* produced by *G*, and a programming language as the set of all strings (programs) producible by *G*

Today

* Hardware generally supports number of Oss and tools, including language interpreters and compilers
* Software holding back hardware

Paradigms

* Procedural / imperative (eg C, Java)
* Functional (or applicative)
* Declarative
* Object-oriented
* Rule-based
* Event-driven
* Parallel / concurrent

Syntax

* Defines allowable arrangement of aymbols, especially fragments
* May constrain fragements to have matching parentheses eg
* Catalogs baseid elements, most importantly the structure

API

* In many cases, ‘real’ programming means writing software to a existing code

Reading code

* Software devs spend a significant fraction time reading, understanding, and modifying existing code
* Emerging notion is that learning how to produce code is based upon reading good code
* Lots of open source code corresponding to popular programs such as browsers, graphics programs, interpreters / compilers, and entire operating systems available for reading and review

Tools

* Editors
  + Syntax highlighting
  + Support language features (matching parens)
* Diff
* Indent

Programs are strings

* Disregard white space

Point(s)

* Source code is sequence of symbols characterized via formal grammars
* Programming language which allows production of these strings exists to facilitiate communication between programmer and compiler

Compiler

* Part is a grammatical recognizer / parser
* Compiler / interpreter for a HLL attempts to generate machine instrutions by determining desired structure of input high level language through parsing input or source code
* Compiler / interpreter begins with processing of program as input string
* Processing based upon checking if string meets a formal specification of the programming language syntax

Alphabet and forming strings

* V
  + = {a, b, c, . . . z}
* Finite, nonempty set of symbols
* Concetenation of a and b, denoted a o b, produces a sequence of two symbols denoeted herafter as ab

Closer set

* Define V+ as {V u V2 u V3 u …}
* V+ is set of all nonempty sentences producible using V
* Adding empty string to V+ produces V\* = {e} o V+
* V\* is denoted the cluser set of V and V+ = V\* - {e} is often called positive of closure of V

Languages using strings

* Grammars used with V to give meaning to subset of strings, L \_c V\*
* L is called a language
  + Languages generated by grammars
  + Viewpoint is that a grammar restricts production of stringsfrom V
  + Grammars used to recognize (parase) elements of a language

Grammar: Formal definition

* G = (Vt, Vn, P, S)
* Consists of following four entities:
  + Set of terminal or primitive symbols, denoted Vt
    - Elemental building blocks of grammar
  + Set of non-terminal symbols, or variables, denoted Vn, or N
    - Used as intermediate quantities in generation of outcomes consistent solely of terminal symbols
  + Set of productions, or rewriting rules, denoted P
* May be used in one of two modes
  + Generative: Used to create string of terminal symbols using P; a sentence in language is generated
  + Analytic: Given sentence, together witspecification of G, one seeks to determine:
    - If sentence was generated by G, and if so
    - Structure of sentence – then we consider semantics

Language: Formal definition

* Any subset L ⊆ is a language
* If |L| is finite, language is called finite, otherwise infinite
* Language generated by grammar G, denoted L(G) is set of all string which satisfy:
  + Each string consists solely of terminal symbols from Vt of G; and
  + Each string was produced from S using P of G

Grammar Types

* Symbols beginning with a capital letter (eg S1 or S) are elements of Vn
* Symbols beginning with lowercase letter (a or b) are elements of Vt
* N denotes length of string s
* Greek letters represent (possibly empty) strings, typically comprised of terminals and/or nonterminals

T1: Context-sensitive

* A t1 or context sensitive grammar restricts productions to:
* α αi β → α βi β
* Meaning βi replaces αi in the context of α and β where α, β ∈ (Vn U Vt)\*, αi

T2: Context free

* In a T2 or context-free grammar, production restrictions are
  + α 1 = S1 ∈ Vn
    - α1 must be single nonterminal
  + S1 → β2
  + Where β2 ∈ (Vn U Vt)\*
* Note a CFG production allows S1 to be replaced by string β2 independently or irrespective of context in which S1 appears

Representaiton and Complexity

* In progressing from a T0 to a T4 grammar, notice that as production restrictions increase, rppreperentational power decreases
* AT same time, increasing production restrictions leads to simpler parser
* Parsing complexity in a T2 grammar is a linear function of the number of rewrite rules in derivation
* Denoting L(Ti) as class of language cgenerated by grammar Ti, above restrictions iicate
* L(T3) c L(T2) < c LT(t1) C L(t0)