Module 01
Languages and Grammar

CS 332 Organization of Programming Languages
Embry-Riddle Aeronautical University
Daytona Beach, FL

M01 Outcomes

At the end of this module you should be able to …

1. State the definition of a symbol, alphabet, string, and language.
2. Justify whether a given example is or is not a language.
3. State the definition of a grammar.
4. Given a description of a language, write its Backus-Naur Form (BNF) or Extended BNF (EBNF).
5. State the four levels of the Chomsky Hierarchy.
6. State the properties required of a grammar for its language to be in any Chomsky level.
7. Given a grammar, place it in the Chomsky hierarchy with justification
8. Given a BNF grammar and a string, provide a derivation for the string.
9. Use regular expressions to define a language.

M01 Language Definition

- A symbol is a single, distinct mark or character.
  - Typically use 1 and 0, or a, b, and c in formal languages.
  - Any symbol applies: λ, β, ✓, ©, », and so on …

- Symbols have no inherent meaning. We supply meaning.

- We've define in terms of written symbols – others acceptable?
  - Spoken words and other sounds (clapping)
  - Gestures, facial expressions, winks, nods, etc
  - Rationale for acceptance: All of these can be transcribed to written symbols

M01 Language Definition

- An alphabet, Σ, is a finite set of symbols.
  - $|\Sigma|$ represents the size of alphabet Σ.
  - If $\Sigma = \{a, b, c\}$, then $|\Sigma| = 3$.

- A string, u, is a sequence of symbols from some Σ.
  - Typically use u, v, and w.
  - λ represents the empty string: a string of length zero having no symbols.
  - The length of string u, $|u|$, is the number of symbols that make it up.
  - If $u = abbca$, then $|u| = 5$; $|\lambda| = 0$.
  - Σ^* represents all possible strings that can be made from Σ.

Note the use of set notation. Not an accident! Sets underlie everything.

M01 Language Definition

- A symbol is a single, distinct mark or character used as a representation.
- An alphabet, Σ, is a finite set of symbols.
- A string, u, is a sequence of symbols from some Σ.
- A language, L, is a (potentially infinite) set of strings.
  - Languages have no inherent meaning.
  - Languages are not about communication. Some are used for communication.
  - Languages are not about programming. Some are used for programming.

M01 Grammar Definition

- A grammar, G, supplies the rules by which a language is constructed
  - We're assuming there is a pattern or structure to the language
  - A set of random strings is still a language
  - Therefore, grammars make sense only for non-random languages

- Backus-Naur Form (BNF) is widely used, universally in Comp Sci
  - BNF uses <u>production rules</u> composed of a left hand side (LHS), and a right hand side (RHS), each containing symbols.
  - The symbols on left hand side (LHS) can be replaced by those on the right hand side (RHS).
  - Production rules take the form LHS → RHS (some authors use LHS := RHS).
  - Example: If you have a rule that says A → aCa, and the string abAca, you can use the rule to replace the "A" in the string with "aCa" – result is ab<u>aCa</u>ca (underlined for visibility).

M01 BNF Grammars

- Elements of a BNF grammar. BNF grammars contain ....
  - Terminals: Symbols in the alphabet, Σ, being used. Convention is lower case.
  - Non-Terminals: Intermediary symbols used in G. Convention is upper case.
  - Start symbol: A special non-terminal indicating the first rule that must be used. This course will use S (Note: Capital S because it's a non-terminal.)

- Sample grammar, G_0, with $\Sigma = \{a, b\}$
  1. S → AB
  2. A → aA
  3. A → a
  4. B → bB
  5. B → b

What strings does G_0 produce?
Strings having at least one a,
and at least one b,
where all a's precede all b's.

That's the language defined by G_0.

M01 Derivation (using the grammar)

- Sample grammar, G_0, with $\Sigma = \{a, b\}$
  1. $S \rightarrow AB$
  2. $A \rightarrow aA$
  3. $A \rightarrow a$
  4. $B \rightarrow bB$
  5. $B \rightarrow b$

Derive u = abbb	
String	Rule
S	
AB	1
AbB	4
AbbB	4
Abbb	5
abbb	3

Derive u = abbb	
String	Rule
S	
AB	1
aB	3
abB	4
abbB	4
abbb	5

Derive u = aabb	
String	Rule
S	
AB	1
aAB	2
aaB	3
aabB	4
aabb	5

Two derivations for the same string. That's typical.

M01 Example #1

- Let $\Sigma = \{a, b\}$
- Write a grammar for language L1, composed of strings ending in bb.
- Provide a derivation for string bb.
- Provide a derivation for string ababbb.

M01 Example Solution

- Let $\Sigma = \{a, b\}$
- Write a grammar for language L_1, composed of strings ending in bb.
- Provide a derivation for string bb.
- Provide a derivation for string ababbb.
- Solution – G_1
    1. $S \rightarrow A$
    2. $A \rightarrow aA$
    3. $A \rightarrow bA$
    4. $A \rightarrow bb$

Is this the only grammar for L_1?

Derive u = bb	
String	Rule
S	
A	1
bb	4

Derive u = ababbb	
String	Rule
S	
A	1
aA	2
abA	3
abaA	2
ababA	3
ababbb	4

Are these the only derivations for these strings?

M01 The Chomsky Hierarchy

- Noam Chomsky, a linguist [1], created a hierarchy of languages that became the foundation of formal language theory
  - This grouping of languages is directly tied to theoretical models of computation.
  - This grouping of languages is directly tied to theoretical limits of computation.
  - This grouping of languages is directly tied to programming languages and compilers
  - This grouping of languages is directly tied to data structures and algorithms

[1] Linguist, philosopher, historian, and more … https://en.wikipedia.org/wiki/Noam_Chomsky

M01 The Chomsky Hierarchy

- For now, define the Chomsky Hierarchy in terms of grammars
- Will describe restrictions on the Left Hand Side (LHS) and Right Hand Side (RHS)
- “Unrestricted” is sometimes called “recursively enumerable” but not covering that in this course – it would take several weeks.

Type	Name	Characteristics of Grammar
Type 3	Regular	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease. Strings derived from right to left, or left to right.
Type 2	Context Free	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease.
Type 1	Context Sensitive	LHS may have terminals and non-terminals. Number of terminals in RHS cannot decrease.
Type 0	Unrestricted	No restrictions

Regular Expressions Introduced

- Our definitions for example languages have been informal so far.
- This is a course about formal systems and formal notation.
- Every regular language can be expressed by a *regular expression*.
  - We'll borrow similar notation for context free languages.
- Trivia: All languages of finite size are regular, because you can always brute force a FSM to recognize it. The FSM might be huge, but it's still finite!

Regular Expression (Informally) Defined

- The set of regular expressions, RE, is recursively defined as follows:
  - λ is in RE.
  - Any single symbol is in RE. True for all symbols in the alphabet, Σ.
  - Concatenation: If u and v are in RE, the uv is in RE.
  - Repetition: If u is in RE, then zero or more copies of u is also in RE. This is written as u*, where the * is known as the Kleene star.
  - Repetition: If u is in RE, then one or more copies of u is also in RE. This is written as u^+.
  - The "or" operator: if u and v are in RE, then the choice of u or v is in RE. This is written as (u + v)

Regular Expression (RE) (Informally) Defined

- What does the previous slide mean?
  - λ is a regular expression.
  - Any single symbol in Σ is a RE.
    - if Σ = {*a, b*}, then *a* is a regular expression and so is *b*.
  - Concatenation: If u and v are in RE, the uv is in RE.
    - if *abba* and *bbab* are regular expressions (they are), then so is *abbabbab*.
  - Repetition: If u is in RE, then zero or more copies of u is also in RE.
    - This is written as u*, where the * is known as the Kleene star.
    - *a** means zero or more copies of *a, and (abb)** means zero or more copies of *abb.*
  - Repetition: If u is in RE, then one or more copies of u is also in RE.
    - This is written as u^+.
    - a^+ means one or more copies of *a, and* $(abb)^+$ means one or more copies of *abb.*
  - The "or" operator: if u and v are in RE, then the choice of u or v is in RE.
    - This is written as (u + v) – an unfortunate overloading of the + operator.
    - (*a* + *b*) means you get to choose *a* or *b*.
    - (*a* + *b*)* means multiple choices of *a* or *b* -- or any string you want.

Regular Expression (Formally) Defined

- The set of regular expressions, RE, is recursively defined as follows:

$$RE = \begin{cases} \lambda \\ a, \forall a \in \Sigma \\ uv, \forall u, v \in RE \\ u*, \forall u \in RE \\ u^+, \forall u \in RE \\ u + v, \forall u, v \in RE \end{cases}$$

Regular Expression Examples

- Let $\Sigma = \{a, b\}$ (For all examples unless stated otherwise)

- The set of all possible strings for an alphabet is written as Σ^*.
  - A very boring language.
- Let L_1 = All strings having 1 or more 'a.'
  - $RE_1 = b^*a(a+b)^*$
  - "Zero or more *b*'s followed by at least one *a*, followed by any sequence of *a*'s and *b*'s."
- Let L_2 = All strings that start with two 'a's.
  - $RE_1 = aa(a+b)^*$
  - "Two *a*'s followed by any sequence of *a*'s and *b*'s."

Regular Expression Examples

- Let $\Sigma = \{a, b\}$ (For all examples unless stated otherwise)
- Let L_3 = Strings that have an even number of 'b's (including λ).
  - $RE_3 = (a^*ba^*b)^*a^*$ -- the *b*'s always show up in groups of two.

- Let L_4 = All strings where no two 'a's are adjacent, including λ.
  - $RE_4 = b^* (ab^+ab^+)^* b^*$

- Let L_5 = All strings with at least three 'b's.
  - $RE_5 = (a^*ba^*ba^*b)\ (a+ b)^*$ -- the + ensures that at least three *b*'s are present.