ITS64304 Theory of Computation

School of Computer Science
Taylor's University Lakeside Campus

Lecture 1: Language and Grammar

Dr Raja..

Learning outcomes

At the end of this topic students should be able to:

- define a formal language for a given grammar*
- Write a regular grammar for a given regular expression or language*

* Aligns to Module Learning Outcome 1 (MLO1)

Languages

- *A language is a set of strings*

- **Definition 1: An alphabet is a finite set of symbols.**
- Examples:
  - Roman: {a, b, c, d, e, f, ... z}
  - Greek: {α, β, γ, δ...}
  - Binary: {0,1}
  - Numeric: {0,1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Alphanumeric: {a-z, A-Z, 0-9}
  - C Tokens

Languages

- *A string is a set of symbols*
- Definition 2: A string over an alphabet ∑ is a finite sequence of symbols from ∑

e.g:

- *watermelon* and *banana* are strings over {a, b, c, d, e, f, ... z}
- *1011010111* and *110* are strings over {0,1}
- *if ((x += 1) >= y) while z* is a string of C tokens
- w_1 = "(3+2) * (9-7)" and w_2 = "72) + 3(*" are strings over ∑ where, ∑ = {0,1,…9, (,), +, -, *, =} for basic arithmetic language
- w_1 is in the language of arithmetic, but w_2 is not.

How to determine this

Languages

- Strings can be empty; we denote the empty string by λ
- The set of all strings over the alphabet $\sum$ (including λ) is denoted by $\sum^*$
- $\sum^*$ represents all possible strings, some of which may not make sense
- A language places restrictions on what set of strings are valid (or legal)
  - For example, this sentence is not a valid English sentence,
    - '*although almost is it*'.
    - "Grave danger you are in. Impatient you are"

Example

- Let $\Sigma = \{a, b, c\}$. The elements of Σ^* include
  - Length 0: λ
  - Length 1: *a b c*
  - Length 2: *aa ab ac ba bb bc ca cb cc*
  - Length 3: aaa aab aac aba abb abc aca …

Kleene star *

Stephen Cole Kleene

Let X be a set. Then

$$X^* = \bigcup_{i=0}^{\infty} X^i \qquad X^+ = \bigcup_{i=1}^{\infty} X^i$$

X^* = set contains all strings that can be built from the elements of X

X^+ = set of all non-null strings over X = XX^*

Languages

- The syntax of programming languages places restrictions on the ordering of constructs

- Natural languages can be very difficult to get right:
  - incorrect syntax: An arrow like flies time
  - correct syntax: An arrow flies like time
  - sensible semantics: Time flies like an arrow
  - sensible semantics (?): fruit flies like a banana

Languages

- **Definition 3: A language over an alphabet $\sum$ is a subset of $\sum^*$**
- Hence a language is just a "certain class" of strings over $\sum$
  e.g.:
  - $\sum = \{0, 1\}$, $L = \{0, 01, 011, 0111, 01111, ...\}$
  - $\sum = \{a,...z\}$, $L = \{ab,cd,efghi,s,z\}$
  - $\sum = \{0, 1\}$, L = { (representations of) primes}
  - $\sum$ = C Tokens, L = { legal C programs }
  - $\sum = \{0, 1\}$, L = {strings containing at least 2 0's}
  - $\sum = \{a,...z\}$, $L = \emptyset$

- In general, the following format is used to specify a language:
  - $L = \{w \in \sum^* \mid w \text{ has property } P\}$
- Hence to define a language, two elements needed:
  - building blocks/alphabets and
  - rules for correct sequence of letters from alphabet

Specifying languages

Several approaches

- Strings matched by a particular regular expressions
  - Sequence or concatenation e.g., ‘0’ followed by ‘1’ (w_1 w_2)
  - Selection or alternation e.g., ‘either 00 or 11’ (w_1 | w_2)
  - Repetition w* e.g., (01)*

    (w zero or more times called Kleene star))
- Strings generated by some rules in a formal grammar
  - A sentence contains a subject followed by a verb phrase
  - <sentence> <noun_phrase> <predicate>
  - E.g., *David went home*
- Strings accepted by some automaton
- Strings for which some YES/NO algorithm output’s “YES”

Describing a Languages

- We need to have some mechanism to describe what are the valid strings within a language.
- Consider the ”language” of correct mathematical expressions (infix notation), involving variable names, *, + (,)
- How can we describe legal phrases in this language?
  - Examples of valid strings:
    - *a * (b * c + d)*
    - *a + b*
    - *(c + d)*
  - Examples of invalid strings:
    - * + *a*
    - + *b* + *
    - *(* c d*

Language Rules

Rule form used in the textbook:

- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow (E)$
- $E \rightarrow id$

- BNF form
  - <expr> ::= <expr> + <expr>
| <expr> * <expr>
| (<expr>)
| <id>
  - <id> ::= string

- Other conventions for specifying language rules include:
  - **DTD**'s (Data Type Definitions) for languages (e.g.: HTML)
  - **Regular Expressions**
    - An expression that describes a set of strings
    - simple languages with only union, concatenation, repetition

Language Rules

- ## Regular Expressions
  - ø represents the empty language
  - Hence **regular expressions are strings over the alphabet {**(,), ø, λ **,** U, ***}** U ∑

A set of strings is regular if it can be generated from the empty set, the set containing the null string, and sets containing a single element of alphabet, using union, concatenation and the kleene star operation.

Definition 4:

1. **Ø, λ and each member of ∑ is a regular expression**
2. **If α and β are regular expressions, then so is (αβ) [concatenation]**
3. **If α and β are regular expressions, then so is (α U β) [union]**
4. **If α is a regular expression, then so is α* [kleene star]**
5. **Nothing else is a regular expression**

- Regular expressions are used as a finite representation of languages
- A language is called regular if it is defined by a regular set

Concatenation of Languages

- Definition: The concatenation of languages X and Y, denoted XY is the language

  $$XY = \{uv \mid u \in X \text{ and } v \in Y\}$$

  The concatenation of X with itself *n* times is denoted X^n. (X length n)

  X^0 is defined as $\{\lambda\}$ (X length 0, hence empty)

- Example:
  - Let X = {a, b, c} and Y = {abb, ba}. Then
    - XY = {aabb, babb, cabb, aba, bba, cba)
    - $X^0 = \{\lambda\}$
    - $X^1 = X = \{a, b, c\}$
    - $X^2 = XX = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
    - $X^3 = X^2X = \{ aaa, aab, aac, aba, abb, abc, aca, acb, \ldots.\}$

Kleene star of Languages

Kleene star of a language, L, denoted by L^*.

L^* is the set of all strings obtained by concatenating zero or more strings from L

Example:

$L = \{a\}$

$L^* = \{e, a, aa, aaa, aaaa, \ldots\}$

$L = \{a, bb\}$

$L^* = \{e, a, bb, aa, bbbb, abb, bba, aabb, abbbb, bbabb, bbaa, bbbbaa\ldots\}$

Language Rules

Definition 5:

- The language L(a) of a regular expression is defined as:

1. $L(\emptyset) = \emptyset$. $L(a) = \{a\}$ for each $a \in \Sigma$
2. If α and β are regular expressions, then $L(\alpha\beta) = L(\alpha)L(\beta)$.
3. If α and β are regular expressions, then $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
4. If α is a regular expression, then $L(\alpha^*) = L(\alpha)^*$.

- $L((a \cup b)^*a) = \{a, aa, ba, aaa, aba, baa, bba, \ldots\}$
  $= \{w \in \{a, b\}^* \mid w \text{ ends in } a\}$
- $L((01 \cup 1)^*) = \{1, 01, 011, 0101, 01011, \ldots\}$
  $= \{w \mid w \text{ does not contain } 00\}$
- $L(0^*(10^*)^*) = \{0, 1, 00, 01, 10, 11, 001, \ldots\}$
  $= \{0, 1\}^*$

Language Rules

- There can be many different regular expressions for a given language

- For example,
  (0 U 1)*
  = ((0 U 1)*)*
  = (0 U 1)*(0 U 1)*
  = (0* U 1*)*

Different regular expressions for the same language

Language Rules

- We often want the "simplest" expression to represent languages
  - (usually minimal number of nested Kleene stars)

- Used in search facilities (vi editor, emacs, grep, egrep, fgrep, . . .) and in compilers
- There are languages that cannot be defined by any regular expression
- for example, there is no regular expression for $L = \{0^n1^n \mid n \geq 0\}$

Formalism to specify languages

- Many formalisms to specify languages
  - Regular expressions, grammars, automata..

- Language: a set of words / strings from a known alphabet
- Need a way to specify correct subset of strings
  - Regular expression is one way where a string must match a regular expression
  - Grammar is another, where a string must be generated from rules
  - Others…. Such as automata

Generating Language Strings

- Consider the language defined by a(a* ∪ b*)b

- First output ‘a’.
- Then output string of ‘a’’s or string of ‘b’’s.
- Then output ‘b’

- Let S = a string, M = the “middle part”, A = a string of a’s, B = a string of b’s
- $S \rightarrow aMb$ (1)
- $M \rightarrow A$ (2)
- $M \rightarrow B$ (3)
- $A \rightarrow aA$ (4)
- $A \rightarrow \lambda$ (5)
- $B \rightarrow bB$ (6)
- $B \rightarrow \lambda$ (7)

To generate the string aaab:

S	
aMb	rule (1)
aAb	rule (2)
aaAb	rule (4)
aaaAb	rule (4)
aaab	rule (5)

Context Free Grammars

- Context free grammars: Rules can be applied to symbols (e.g. A) regardless of context of symbol

- This makes them computationally useful

- e.g: Even length strings over {a,b}

  $$S \rightarrow aO \mid bO \mid e$$

  $$O \rightarrow aS \mid bS$$

$$S \Rightarrow aO \Rightarrow abS \Rightarrow abbO \Rightarrow abbbS \Rightarrow abbb$$

Context Free Grammars

- Strings with an even number of b's (i.e. a*(ba*ba*)*)

  $S \rightarrow aS \mid bB \mid \lambda$

  $B \rightarrow aB \mid bS$

  $S \Rightarrow aS \Rightarrow abB \Rightarrow abbS \Rightarrow abbbB \Rightarrow abbbaB \Rightarrow$
  $abbbabS \Rightarrow abbbab$

- Strings not containing abc

  $S \rightarrow aB \mid bS \mid cS \mid \lambda$

  $B \rightarrow aB \mid bC \mid cS \mid \lambda$

  $C \rightarrow aB \mid bS \mid e$

  $S \Rightarrow aB \Rightarrow abC \Rightarrow abaB \Rightarrow abacS \Rightarrow abac$

Context Free Grammars

- Palindromic strings, $w = w^R$, over $\{a,b\}$

  $S \rightarrow a \mid b \mid \lambda$

  $S \rightarrow aSa \mid bSb$

  $S \Rightarrow aSa \Rightarrow abSba \Rightarrow ababa$

Context Free Grammars

Formal Definition

- A context-free grammar G is a quadruple (V,∑,R,S) where
- V is a finite set of symbols (terminals and non-terminals)
- ∑ is the set of terminal symbols (the symbols of the language)
- S is a distinguished element of V - ∑ called the start symbol, and
- R is a set of rules

- Members of V - ∑ are called non-terminals

- The set of rules, R is a finite subset of
- $V - \Sigma \times V^*$ (from nonterminals to a string of nonterminals and terminals)

Context Free Grammars

e.g:

- Grammar G consists of rules $\{S \rightarrow aSb \mid S \rightarrow \lambda\}$.

- Using these rules we can get:
  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

  The language of G, denoted L(G), is the set $\{w \in \Sigma^* \mid S \Rightarrow w\}$.

- This is clearly $\{a^n b^n : n \geq 0\}$.

- Recall that a regular expression can't specify this language

Parse Trees and Derivations

- Derivations can be written in a graphical form as a parse tree

- Given a grammar and a string, there may be different derivations to get the same string
- Equivalent derivations (same meaning) have the same parse tree
- Any parse tree has unique *leftmost* and *rightmost* derivations
- Grammars with strings having 2 or more parse trees are *ambiguous*
- Some ambiguous grammars can be rewritten as equivalent unambiguous grammars

Parse Trees and Derivations

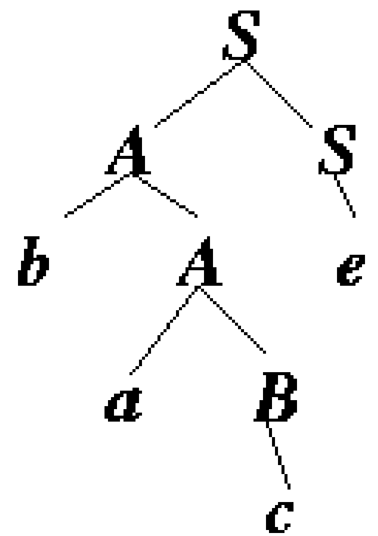
Example Derivation as Parse Tree

- $S \to AS \mid SB \mid \lambda$
- $A \to aB \mid bA \mid \lambda$
- $B \to bS \mid c \mid \lambda$

- $S \Rightarrow AS \Rightarrow bAS \Rightarrow baBS \Rightarrow bacS \Rightarrow bac$

- A parse tree of $S \overset{*}{\Rightarrow} w$ is obtained as follows:

- The parse tree has root S
- If $S \to AS$ is the rule applied to S,
  then add A and S as children of S.
- $A \to bA$, then add b and A as children of A ...
- If $A \to \lambda$ is the rule applied to A,
  then add λ as the only child of A



Equivalent Derivations

- Consider the simple grammar
- G = (V, S,R,S) where V = {S, (,)},
- S = {(,)},R = {S -> SS | (S) | λ}

- The string (())() can be derived from S by several derivations, e.g…

(D1) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$((S))S $\Rightarrow$(())S $\Rightarrow$(())(S) $\Rightarrow$(())()

(D2) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$((S))S $\Rightarrow$((S))(S) $\Rightarrow$(())(S) $\Rightarrow$(())()

(D3) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$((S))S $\Rightarrow$((S))(S) $\Rightarrow$((S))() $\Rightarrow$(())()

(D4) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$((S))(S) $\Rightarrow$(())(S) $\Rightarrow$(())()

(D5) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$((S))(S) $\Rightarrow$((S))() $\Rightarrow$(())()

(D6) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$(S)() $\Rightarrow$((S))() $\Rightarrow$(())()

(D7) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$((S))(S) $\Rightarrow$(())(S) $\Rightarrow$(())()

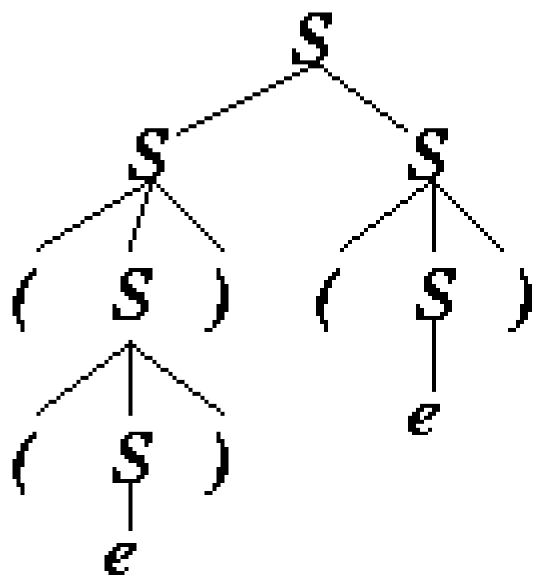
(D8) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$((S))(S) $\Rightarrow$((S))() $\Rightarrow$(())()

(D9) S $\Rightarrow$ SS $\Rightarrow$(S)S $\Rightarrow$(S)(S) $\Rightarrow$(S)(S) $\Rightarrow$((S))() $\Rightarrow$(())()

(D10) S $\Rightarrow$ SS $\Rightarrow$ S(S) $\Rightarrow$ S() $\Rightarrow$(S)() $\Rightarrow$((S))() $\Rightarrow$(())()

Equivalent Derivations

- These all have the same parse tree



Derivations

- Derivations, D and D' are similar if they can be transformed into each other via switching the order in which the rules are applied
- CFG always has 1 *leftmost derivation* and *1 rightmost derivation* (D1 leftmost, D10 rightmost, above)
- To get leftmost derivation, always replace leftmost non-terminal in current string

Ambiguous Grammars

- Recall the grammar G = V, S,R,S where
  V = *, +, (,),E
  S = *, +, (,)
  R = $S \rightarrow E$
  $E \rightarrow E + E \mid E * E \mid (E) \mid id$

- The string id + id * id can be generated by this grammar according to two different parse trees.

- Only one of these (a) corresponds to the "natural" semantics of id + id * id, where * takes precedence over +.

Ambiguous Languages

- Many ambiguous grammars (such as the one on the previous slide) can easily be converted to an unambiguous grammar representing the same language.

- Some context free languages have the property that all grammars that generate them are ambiguous. Such languages are inherently ambiguous.

- Inherently ambiguous languages are not useful for programming languages, formatting languages, or other languages which must be parsed automatically

Parsing

- Given a context-free grammar G and input w determine if $w \in L(G)$.
- How do we determine this for **all** possible strings?
- Multiple derivations may exist
- Must also discover when no derivation exists

- A procedure to perform this function is called a parsing algorithm or parser.

- Some grammars allow deterministic parsing, i.e. each sentential form has at most one derivation

Ambiguity and Parsing

- A grammar is unambiguous if at each step in a leftmost derivation there is only one rule which can lead to the desired string.
- Deterministic parsing is based upon determining which rule to apply.

Conclusion

- Grammars are simple rules
- Simple for machines to process
- Need to avoid ambiguous grammars

Q&A

Read your lesson materials….

Next: Attempt Tutorial 1