System programming

Lecture 4

Fundamentals of Language Specification

- A specification of the source language forms the basis of source program analysis

Programming Language Grammars – 1/2

- The lexical and syntactic features of a programming language are specified by its grammars
- A language L can be considered to be a collection of valid sentences
- Each sentence can be looked upon as a sequence of words, and each word as a sequence of letters or graphic symbols acceptable in L

Programming Language Grammars – 2/2

- A language specified in this manner is called ***formal language***
- A formal language grammar is a set of rules which precisely specify the sentences of L
- Natural languages are not formal languages due to their rich vocabulary
- **PLs are formal languages**

Terminal symbols, alphabet and strings – 1/5

- The alphabet of L, denoted by the Greek symbol Σ, is the collection of symbols in its character set
- Use lower case letters a, b, c, etc. to denote symbols in Σ
- ***A symbol in alphabet is known as a terminal symbol (T) of L***
- The alphabet can be represented using the mathematical notation of a set, e.g.

$$\Sigma \equiv \{a, b, \ldots z, 0, 1, 2, \ldots 9\}$$

Terminal symbols, alphabet and strings – 2/5

- The symbols {, ‘,’ and } are part of the notation
- We call them **metasymbols** to differentiate them from terminal symbols
- Assume metasymbols are distinct from the terminal symbols
- If they are not, we **enclose the terminal symbol in quotes to differentiate it from metasymbol**

Terminal symbols, alphabet and strings – 3/5

- Eg. Set of punctuation symbols of English can be defined as

{:, ;, ‘,’, …}

where ‘,’ denotes the terminal symbol comma

Terminal symbols, alphabet and strings – 4/5

- **A string is a finite sequence of symbols**.
- Represent strings by Greek symbols α, β, γ, etc
- Thus α = axy is a string over Σ

- ***Length of the string is the number of symbols in it***

- Absence of any symbol is also a string, the **null string** ε

- The **concatenation** operation combines two strings into a single string

Terminal symbols, alphabet and strings – 5/5

- Given two strings α and β, concatenation of α with β yields a string which is formed by putting the sequence of symbols forming α before the sequence of symbols forming β

- Example : If α = ab, β = axy, then concatenation of α and β, represented as $\alpha.\beta$ or simply $\alpha\beta$, gives the string abaxy

- The null string can also participate in a concatenation

- Thus $\mathbf{a.\ \varepsilon = \varepsilon.a = a}$

Nonterminal symbols

- **A nonterminal symbol (NT) is the name of a syntax category of a language**, e.g noun, verb, etc
- NT is written as a single capital letter, or as a name enclosed between <…>, eg. A or <Noun>
- During grammatical analysis, a nonterminal symbol represents an instance of the category
- Thus, *<Noun> represents a noun*

Productions – 1/5

- **A production, also called a rewriting rule, is a rule of the grammar**

  A nonterminal symbol ::= String of Ts and NTs

- It defines the fact that the NT on LHS of production can be rewritten as the string of Ts and NTs appearing on RHS

- When an NT can be written as one of many different strings, the symbol '|' (standing for or) is used to separate the string on RHS

Productions – 2/5

- Eg.

  <Article> ::= a | an | the

- **The string on RHS of production can be a concatenation of component strings**

- Eg. The production

  <Noun Phrase> ::= <Article> <Noun>

  expresses the fact that the noun phrase consists of an article followed by a noun

- Each grammar G defines a language L_G
- **G contains NT called distinguished symbol or start NT of G**

Productions – 3/5

- Unless, otherwise specified, we **use the symbol S as the distinguished symbol of G**

- A valid string α of L_G is obtained by using the following procedure
  1. Let α = 'S'
  2. While α is not a string of terminal symbols

     (a) Select an NT appearing in α, say X

     (b) Replace X by a string appearing on RHS of a production of X

Productions – 4/5

- Eg.1.13: Grammar defines a language consisting of noun phrases in English

<Noun Phrase>	::=	<Article> <Noun>
<Article>	::=	a | an | the
<Noun>	::=	boy | apple

<Noun Phrase> is distinguished symbol of grammar

The boy and an apple are some valid strings in the language

Productions – 5/5

- A grammar G of a language L_G is a quadruple (Σ, SNT, S, P) where

  where Σ is the alphabet of LG i.e. set of Ts

  SNT is the set of NTs

  S is distinguished symbol

  P is the set of productions

Derivation, reduction and parse trees

- A grammar G is used for **two purposes**
  - To **generate valid strings** of L_G → **derivation**
  - To **'recognize' valid strings** of L → **reduction**

- A parse tree is used to depict the syntactic structure of a valid string as it emerges during a sequence of derivations or reductions

Derivation - 1/6

- Let production P_1 of grammar G be of the form

  $P_1 : A ::= \alpha$

  and let β be a string such that $\beta \equiv \gamma A \theta$, then replacement of A by α in string β constitutes a derivation according to P_1

Derivation - 2/6

- Use the notation $N \Rightarrow \eta$ to denote direct derivation of η from N and $N \Rightarrow^* \eta$ to denote transitive derivation of η (i.e. derivation in zero or more steps) from N, respectively

Derivation - 3/6

- Thus, $A \Rightarrow \alpha$ only if $A ::= \alpha$ is a production of G and $A \Rightarrow \delta$ if $A \Rightarrow \ldots. \Rightarrow^* \delta$

- We can use this notation to define a valid string according to a grammar G as follows:

  δ is a valid string according to G only if

  $S \Rightarrow^* \delta$, where S is distinguished symbol of G

Derivation - 4/6

- Eg.1.14: Derivation of the string the boy according to grammar can be depicted as

$$
\begin{aligned}
\langle\text{Noun Phrase}\rangle &\Rightarrow \langle\text{Article}\rangle\ \langle\text{Noun}\rangle \\
&\Rightarrow \text{the}\ \langle\text{Noun}\rangle \\
&\Rightarrow \text{the boy}
\end{aligned}
$$

- A string α such that $S \Rightarrow^* \alpha$ is a sentential form of L_G
- The string α is a sentence of L_G if it consists of only Ts

Derivation - 5/6

- Eg.1.15 : consider the grammar

<Sentence>	::= <Noun Phrase> <Verb Phrase>
<Noun Phrase>	::= <Article> <Noun>
<Verb Phrase>	::= <Verb> <Noun Phrase>
<Article>	::= a | an | the
<Noun>	::= boy | apple
<Verb>	::= ate

Derivation - 6/6

- The following strings are sentential forms of L_G

<Noun Phrase> <Verb Phrase>

the boy <Verb Phrase>

<Noun Phrase> ate <Noun Phrase>

the boy ate <Noun Phrase>

the boy ate an apple

However, only **the boy ate an apple** is a sentence

Reduction - 1/3

- Let production P_1 of grammar G be of the form

  $P_1 : A ::= \alpha$

  and let σ be a string s.t. $\sigma \equiv \gamma\alpha\theta$, then replacement of α by A in string σ constitutes a reduction according to production P_1

- Use the notations $\eta \rightarrow N$ and $\eta \rightarrow^* N$ to depict direct and transitive reduction, resp

- Thus, $\alpha \rightarrow A$ only if $A ::= \alpha$ is a production of G and $\alpha \rightarrow A$ if $\alpha \rightarrow \ldots \rightarrow^* A$

Reduction - 2/3

- We define the validity of some string δ according to grammar G as follows :

  δ is a valid string of L_G if $\delta \rightarrow^* S$, where S is distinguished symbol of G

- Eg.1.16 : To determine the validity of string

  the boy ate an apple

  According to grammar we perform the following reductions

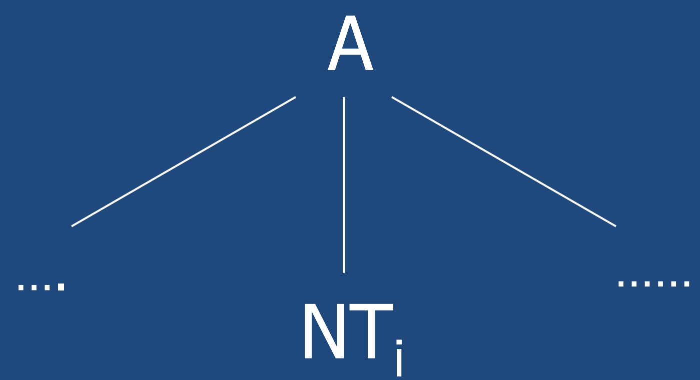
Reduction - 3/3

Step	String
0	the boy ate an apple
1	<Article> boy ate an apple
2	<Article> <Noun> ate an apple
3	<Article> <Noun> <Verb> an apple
4	<Article> <Noun> <Verb> <Article> apple
5	<Article> <Noun> <Verb> <Article> <Noun>
6	<Noun Phrase> <Verb> <Article> <Noun>
7	<Noun Phrase> <Verb> <Noun Phrase>
8	<Noun Phrase> <Verb Phrase>
9	<Sentence>

the boy ate an apple → *<Sentence>

Parse trees – 1/3

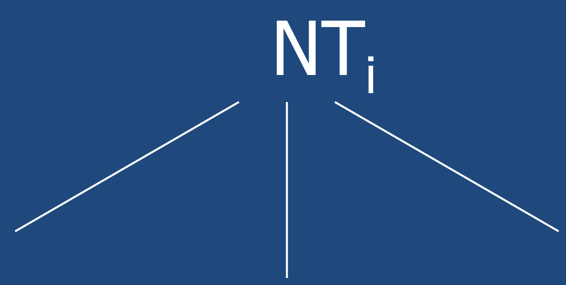
- A sequence of derivations or reductions reveals the syntactic structure of a string w.r.t. G
- We depict the syntactic structure in the form of a parse tree
- Derivation according to production A ::= α gives rise to following elemental parse tree



(Sequence of Ts and NTs constituting α)

Parse trees – 2/3

- A subsequent step in the derivation replaces an NT in α, say NT_i, by a string
- We can build another elemental parse tree to depict this derivation



Parse trees – 3/3

- We can combine the two trees by replacing the node of NT_i in first tree by this tree
- The parse tree has grown in downward direction due to derivation
- Obtain a parse tree from a sequence of reductions by performing the converse actions
- Such a tree would grow in **upward direction**

Classification of grammars – 1/5

- Grammars are classified based on the nature of productions used in them
- Type-0 grammars
  - Known as phrase structure grammars
  - Contain productions of the form

$$\alpha ::= \beta$$

  where both α and β can be strings of Ts and NTs
  - Such productions permit arbitrary substitution of strings during derivation or reduction
  - Hence they are not relevant to specification of programming languages

Classification of grammars – 2/5

- Type-1 grammars
  - Known as context sensitive grammars
  - Production specify that derivation or reduction of strings can take place only in specific contexts
  - A Type-1 production has the form

    $\alpha\ A\ \beta ::= \alpha\ \pi\ \beta$
  - Thus a string π in a sentential form can be replaced by 'A' (or vice versa) only when it is enclosed by the strings α and β
  - These grammars are also not particularly relevant for PL specification since recognition of PL constructs is not context sensitive in nature

Classification of grammars – 3/5

- Type-2 Grammars
  - Impose no context requirements on derivations or reductions
  - It has the form

    $$A ::= \pi$$

    which can be applied independent of its context
  - Known as context free grammars (CFG)
  - CFGs are ideally suited for programming language specification

Classification of grammars – 4/5

- Type-3 Grammars
  - Characterized by the production

    A ::= tB | t   or

    A ::= t | tB
  - These productions also satisfy Type-2 grammars
  - Use of Type-3 specification is restricted to specification of lexical units, eg. Identifiers, constants, labels, etc
  - Productions for <constant> and <identifier> in grammar (1.3) are in fact Type-3 in nature

Classification of grammars – 5/5

- This can be clearly seen when we rewrite the production for <id> in the form Bt | t, as

  <id> ::= l | <id> l | <id> d

  where l and d stands for letter and digit resp.
- They are also known as linear grammars or **regular grammars**
- Further categorized into left-linear and right-linear grammars depending on whether NT in RHS alternative appears at extreme left or extreme right