CPSC320

Tutorials

Robert Pringle

September 17, 2007

Logic review.

For the following assume symbols p and q are generic propositions, x is an arbitrary entity and p_x is proposition function relying on x.

Symbol	Example	Meaning
$\forall$	$\forall x\ p_x$	For all instances of x p_x is true.
$\exists$	$\exists x\ p_x$	There exists an instance of x for which p_x is true.
$\neg$	$\neg p$	Negation of the proposition p.
$\wedge$	$p \wedge q$	And statement that is true if both p and q are true.
$\vee$	$p \vee q$	Or statement that is true if either p, q or both are true.
$\mid$	$p \mid q$	p is true if q is true.

String review.

For the following assume that x and y are generic strings and a and b are generic symbols.

Operation/Entity	Example	Meaning
Concatenation	xy	xy is the concatenation of strings x and y. Note that xy is not the same as yx.
Alphabet	$\{a, b\}$	An alphabet is a finite non-empty set of symbols.
String	abbaa for $\{a, b\}$	A sequence of zero or more elements from an alphabet.
Length	$\|\text{abba}\| = 4$	The number of symbols present in a string.
Empty String	ϵ	The string with no symbols. By definition: $\epsilon x = x\epsilon = x$ $\{\epsilon\}A = A\{\epsilon\} = A$

Set notation review.

For the following assume symbols A and B are generic sets, x and y are generic symbols, and that n is a positive integer.

Operation/Entity	Usage/Symbol	Meaning	Descriptive Meaning
Membership	$x \epsilon A$		x is a member of set A.
Union	$A \cup B$	$x \epsilon A \cup B \mid x \epsilon A \vee x \epsilon B$	$A \cup B$ is the set containing all elements present in sets A or B.
Intersection	$A \cap B$	$x \epsilon A \cap B \mid x \epsilon A \wedge x \epsilon B$	$A \cap B$ is the set containing all elements the are present in both sets A and B.
Subset	$A \subseteq B$	$\forall x \epsilon A \; x \epsilon B$	A is a subset of B if all elements of A are contained in the set B.
Concatenation	$A \bullet B$ or AB	$xy \epsilon AB \mid x \epsilon A \wedge y \epsilon B$	AB is the set containing all element concatenations between the sets A and B with A's members being the prefix and B's members the postfix.
Cartesian Product	$A \times B$	$\{x, y\} \epsilon A \times B \mid x \epsilon A \wedge y \epsilon B$	$A \times B$ is the set that contains all possible subsets formed from all possible element combinations of both A and B starting with elements from A and ending with elements from B.

Set notation review.

Operation	Usage/Symbol	Meaning	Descriptive Meaning
Power	A^n	$A^n = AA^{n-1}$, $A^2 = AA$, $A^1 = A$ and $A^0 = \epsilon$	A^n is a set formed by applying the concatenation product operation between n instances of A. For example $A^2 = AA$.
Empty Set	ϕ		ϕ is the set containing no elements, which is not to be confused with the set containing the empty string, $\{\varepsilon\}$. By definition: $\phi \cup A = A$ $\phi \cap A = \phi$ $\phi A = A\phi = \phi$ $A \times \phi = \phi \times A = \phi$.
Cardinality	$\|A\|$		$\|A\|$ is the number of elements contained in the set A.
Kleene Operation $+$	A^+	$\bigcup_{n=1}^{\infty} A^n$	A^+ is the containing all powers of a set A excluding A^0.
Kleene Operation $*$	A^*	$\bigcup_{n=0}^{\infty} A^n$	A^* is the containing all powers of a set. Note that: $A^* = A^+ \cup \{\epsilon\}$ and $AA^* = A^*A = A^+$

Language and grammar review.

- A language over an alphabet A is defined as a subset of A^*.
- A formal grammar is used to generate, for a generative grammar, or recognize, for an analytical grammar, strings contained in the language it represents.
- Grammars are usually defined by a 4-tuple containing a set of terminal symbols, a set of non-terminal symbols, a set of productions and the set containing the starting productions for the grammar.

Formal grammar definition example.

Let us consider the language, L, of strings having a length of at least one and containing only 0s. Below is an example grammar for this language.

Let G be the grammar representing the language L and P the set productions for G. Then we have:

$$G = (\{S, F\}, \{0\}, P, \{S\})$$

P: $S \rightarrow 0S$
$S \rightarrow F$
$F \rightarrow 0$

Chomsky Grammar Hierarchy Review.

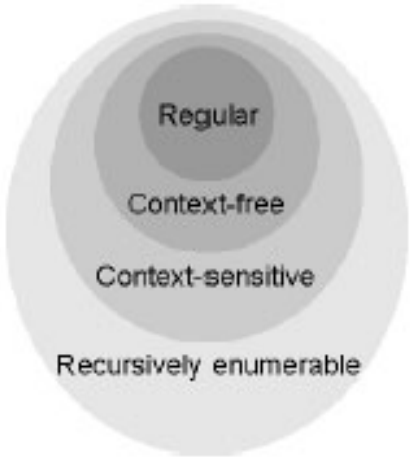


Figure: Chomsky Grammar Hierarchy.

Chomsky Hierarchy Grammar Restrictions.

Grammar Type	Production Forms	Restriction Description
Regular	$N \rightarrow NT^*$, $N \rightarrow T^*N$ or $N \rightarrow T^*$	Regular grammars use productions that only have one non-terminal on the right hand side of the production which is either the first or last symbol for right hand side of the production.
Context-free	$N \rightarrow (N^* T^*)^*$	Context-free grammars restrict the left-hand side of the productions to a single non-terminal symbol. Any combination of terminals and non-terminals is acceptable on the right hand side.
Context-sensitive	$X \rightarrow Y$ with $\|X\| \leq \|Y\|$ and $X, Y \subseteq (N^* T^*)^*$	Context-sensitive grammars require that rules be non-reducing from left to right (there must always be more terminals on the right-hand side than on the left-hand side).
Recursively Enumerable	$(N^* T^*)^* \rightarrow (N^* T^*)^*$	Recursively enumerable grammars have no restrictions on their production rules.

Chomsky Hierarchy practice questions.

Let us consider a language L having a grammar G with $G = (\{S, A, B\}, \{0, 1\}, P, \{S\})$. For each of the following production definitions describe the lowest level of the chomsky hierarchy to which G belongs.

1. $S \rightarrow 0S$, $S \rightarrow 0A$, $A \rightarrow 1A$, $A \rightarrow 1B$, $B \rightarrow 0$
2. $S \rightarrow 0S$, $0S \rightarrow 1S1$, $A1S \rightarrow 0S0$, $A \rightarrow 1A1$, $A \rightarrow B$, $11B \rightarrow 0$
3. $S \rightarrow 0S0$, $S \rightarrow A$, $A \rightarrow 1A1$, $A \rightarrow B$, $B \rightarrow 0$
4. $S \rightarrow 0S$, $S0A \rightarrow 1S1B$, $BS \rightarrow 1A1B$, $A \rightarrow 0B0$, $B \rightarrow 1$
5. $S \rightarrow 1SA$, $S \rightarrow 1S$, $A \rightarrow 1A1B$, $A \rightarrow 0$, $B \rightarrow 1$

Chomsky Hierarchy practice solutions.

Let us consider a language L having a grammar G with $G = (\{S, A, B\}, \{0, 1\}, P, \{S\})$. For each of the following production definitions describe the lowest level of the chomsky hierarchy to which G belongs.

1. Regular grammar.
2. Recursively Enumerable grammar.
3. Context-free grammar.
4. Context-sensitive grammar.
5. Context-free grammar.

Regular Expressions

- A regular expression is an expression that can be used to generate or recognize strings in a particular language.
- A regular grammar can be equivalently represented by a regular expression.

Regular Expressions Operators

Operation/Entity	Usage/Symbol	Meaning
Alternation	$x\|y$	This regular expression indicates the next symbol generated/recognized will be either x or y.
Grouping	(x)	Similar to arithmetic expression this is used to group portions of the regular expressions together thus changing the precedence and scope of operators applied around them. Consider $x\|y^+$ and $(x\|y)^+$ that represent different languages.
Repetition (0 or more)	x^*	This operator is used to represent a repetition of the symbol/regular expression it is applied to zero or more times.
Repetition(1 or more)	x^+	This operator is used to represent a repetition of the symbol/regular expression it applied to one or more times.
Option	$x?$	This operator is used to represent a symbol/regular expression that is optional, it can be applied one or zero times.

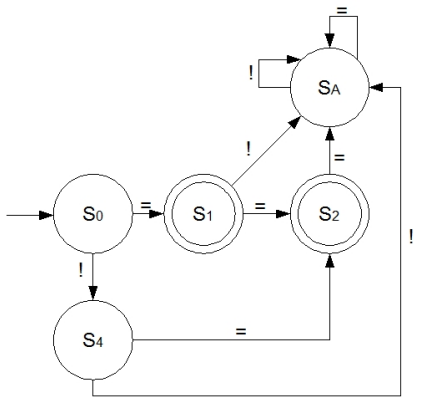
Finite Automata

- A finite automata is another way one can recognize or generate strings in a particular language.
- An automata is represented by a number of states and transitions and can be represented as a table or graphically.
- Transitions in a finite automata are usually triggered by a symbol in the alphabet of the language you are trying to recognize but for nondetermistic finite automata ϵ transitions are also possible.

Finite Automata Example

Consider the following different finite automata recognizing the language containing the $==$, $!=$ and $=$ strings with an alphabet $\{ =, ! \}$. Below is a discrete finite automata represented in both tabular and graphical form for this language.

State		Input	
		$=$	!
$\rightarrow$	S_0	S_1	S_4
*	S_1	S_2	S_A
*	S_2	S_A	S_A
	S_4	S_2	S_A
	S_A	S_A	S_A



Regular grammars.

- A regular grammar can be left-linear or right-linear.
- Left-linear grammars have non-terminals on the leftmost side of the right hand of the production rule.
  - $N \longrightarrow NT^*$ and $N \longrightarrow T^*$
- Right-linear grammars have non-terminals on the rightmost side of the right hand of the production rule.
  - $N \longrightarrow T^*N$ and $N \longrightarrow T^*$

Regular grammar practice

For the following language descriptions define a grammar, G, that generates the language as well as a regular expression and discrete finite automata if the language is regular.

1. A language of 0s and 1s which is prefixed by a string of zero or more zeros and suffixed by one or more 1s.
2. A language consisting of all combinations of the letters a, b, c and d of length 3.
3. A language over the english alphabet recognizing all words starting with a and ending with t.
4. A language for C++ array declarations of type **int**, **bool** and **float** without initialization.

Context-free grammar practice

For the following language descriptions define a grammar, G, that generates the language. If the language can be represented by a regular grammar this should be indicated.

1. A langauge of arithmetic expressions consisting of the binary operations * and / and the operands c and d.
2. A language of 0s and 1s consisting of palindromes.
3. A language of 0s and 1s where the number of 0s and 1s in the string is equivalent.
4. A language over the english alphabet recognizing words that do not start a and end with t and having the character d between the first and last letters of the word.

Grammar Derivations

- A derivation shows the steps taken through the productions of a grammar in recognition or generation of a string.
- Left-most derivations expand productions from the left most non-terminal of the right hand side of a production.
- Right-most derivations expand productions from the right most non-terminal of the right hand side of a production.
- Derivations can also be represented by derivation trees whose root is the first non-terminal use and subsequent non-leaf subtrees the non-terminals used and the left nodes the terminals matched.

Grammar Derivation Example

Consider the grammar, $G = (\{E\}, \{a, b, +, -\}, P, \{E\})$ with the P defined as the following set of productions:

P: $E \rightarrow E + E$
$E \rightarrow E - E$
$E \rightarrow a$
$E \rightarrow b$

For the string a+b-a we have the following derivations:

Leftmost derivation: E $\Rightarrow E + E \Rightarrow a + E \Rightarrow a + E - E \Rightarrow a + b - E \Rightarrow a + b - a$

Rightmost derivation: E $\Rightarrow E + E \Rightarrow E + E - E \Rightarrow E + E - a \Rightarrow E + b - a \Rightarrow a + b - a$

Grammar Derivation Example

For the string a+b-a with the grammar and derivations from the previous slide we have the following derivation tree:

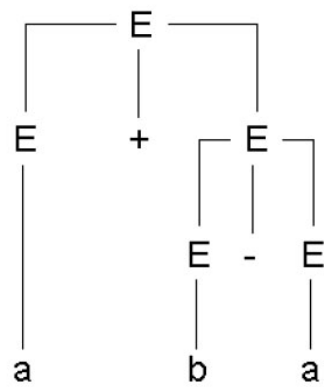


Figure: Example derivation tree.

Ambiguous Grammars

- A grammar is considered ambiguous if there exists a string in the language the grammar represents that has more than one leftmost or rightmost derivation.
- Practice: Would you consider the grammar used in the previous example to be ambiguous? Why or why not?

Backus Naur Form

- A method to represent grammars using meta-symbols.
- Production rules are assigned to non-terminals through the ::= symbol.
- Elements contained in $< >$ are used to represent non-terminals.
- When a non-terminal has more than one production rule the symbol | can be used to show production options for the non-terminal.
- Symbols in the grammar productions that overlap symbols in this form are usually surrounded by single or double quotes to indicate that they are symbols from the language and not meta-symbols.
- Practice: Reform the grammars from the context-free practice to BNFs.

Extended Backus Naur Form

- An extension to BNF to allow for a more condensed representation of a grammar.
- Elements contained in ( ) form a group somewhat like an embedded production rule.
- $\{\ \}^+$ is used to indicate that whatever is contained in the $\{\ \}$ is repeated one or more times.
- $\{\ \}$ or $\{\ \}^*$ is used to indicate that whatever is contained in the $\{\ \}$ is repeated zero or more times.
- [ ] is used indicate that whatever is contained in-between is optional and occurs zero or one times.

Syntax Diagram

- Method to represent grammar diagrammatically.
- Terminals in a syntax diagram are represented by ovals and non-terminals by squares.
- Each path from start to end represents a string generated or recognized by the grammar.
- Production alternatives, BNF $|$, repetition of one or more, EBNF $\{\ \}^{+}$, repetition of zero or more, EBNF $\{\ \}$ and options, EBNF $[\ ]$ can all be easily represented in a syntax diagram.

Syntax Diagram

For the following example consider a grammar with non-terminal A and terminal 0. Below we show the syntax diagram equivalence to various BNF and EBNF productions.

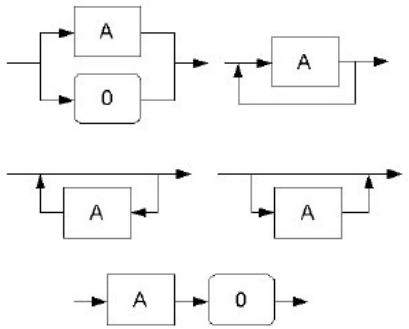


Figure: Syntax Diagram Examples.

Position	Represents	Equivalent BNF or EBNF production
Top left	Alternative	A | 0
Top Right	Repetition of one or more.	$\{ A \}^{+}$
Middle Right	Repetition of zero or more.	$\{ A \}$
Middle Left	Optional	[A]
Bottom	General Production	A 0

Grammar Practice

For each of the following languages define a formal grammar that generates or recognizes strings in the language. After this form an equivalent BNF, EBNF and syntax diagram for the grammar.

1. A conditional statement in the form commonly found in C. Assume that the production for arithmetic expressions is already defined as EXP, the set of terminals this productions uses is defined as as T_{EXP} and non-terminals as N_{EXP}. Also assume that the productions for other C statements is already defined as STMT, the set of terminal this production uses is defined as T_{STMT} and non-terminals as N_{STMT}.
2. Arithmetic expressions on integers allowing the binary operations +,-,* and /.