Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 8: Context-Free grammars and parsing algorithms

John Longley

week 5: Monday 9th - Friday 13th February, 2026

1. Recall the following simple context-free grammar for arithmetic expressions from Lecture 22. The start symbol is Exp.

$$\begin{array}{rcl} \mathsf{Exp} & \to & \mathsf{Var} \mid \mathsf{Num} \mid (\ \mathsf{Exp}\ ) \\ \mathsf{Exp} & \to & \mathsf{Exp} + \mathsf{Exp} \\ \mathsf{Exp} & \to & \mathsf{Exp} * \mathsf{Exp} \\ \mathsf{Var} & \to & x \mid y \mid z \\ \mathsf{Num} & \to & 0 \mid \cdots \mid 9 \end{array}$$

   (a) How many syntax trees are there for each of the following three strings? Draw them all.

$$3 + x * y \qquad 3 + (x * y) \qquad z + 10$$

   (b) Design a new context-free grammar that generates exactly the same language as the one above, but with the property that it is *unambiguous*: every string in the language should have exactly one syntax tree. Informally, your grammar should enforce the familiar convention that * takes precedence over +. You will find it helpful to introduce some additional non-terminal symbols.

   [Hint: First try to do this for the grammar with the rule for $\mathsf{Exp} * \mathsf{Exp}$ omitted. To ensure that a string like $3 + 4 + 5$ has only one tree, you might want to draw inspiration from the grammar for comma-separated lists in Lecture 21. Then try to adapt your grammar to cater for *, building in the precedence rule.]

   (c) For the grammar you have designed in part (b), draw the *unique* syntax tree for any of the strings from part (a) that had more than one syntax tree with respect to the original grammar.

2. Consider the following context free grammar with start symbol S:

S	→	NP VP	PP	→	Pre NP
S	→	*I* VP PP	V	→	*ate*
NP	→	Det N	Det	→	*the* | *a*
VP	→	*ate* NP	N	→	*fork* | *salad*
VP	→	V	Pre	→	*with*

   (a) Convert this grammar to Chomsky Normal Form (see Lecture 23).

   (b) Use the CYK algorithm from Lecture 23 to parse the sentence

   I ate the salad with a fork

   (c) How many complete analyses of the sentence do you get? Draw their syntax trees.

   (d) Now add an extra production rule to the CNF grammar to allow us to parse the overall NP 'the salad with a fork' as a NP followed by a PP.

   Revise your CYK chart or graph to include any new entries this introduces.

   How many complete analyses are there now for the overall sentence "I ate ..."?

3. Consider the following grammar for arithmetic expressions such as (n ∗ n ∗ n). Here n stands for the lexical category of *numeric literals* such as 5 and −23.

Terminals:	(,), ∗, n
Nonterminals:	Exp, Ops
Productions:	Exp → n Ops | (Exp)
	Ops → ϵ | ∗ n Ops
Start symbol:	Exp

   This grammar is somewhat restrictive — for example, it does not admit the string (n ∗ n) ∗ n — but it will do as an example.

   This is in fact an LL(1) grammar with the following parse table:

	(	)	∗	n	$
Exp	(Exp)			n Ops	
Ops		ϵ	∗ n Ops		ϵ

   Here, for example, the top left entry (Exp) stands for the production Exp → (Exp).

   (a) Using this table, apply the LL(1) parsing algorithm from Lecture 24 to the input

   (n ∗ n)

   At each step, show the operation applied, the input string remaining, and the stack state, as in the lecture.

   (b) For each of the following three input strings, explain how and where an error arises in the course of the LL(1) parsing algorithm. In each case, suggest an error message that an LL(1) parser could issue to the author of the input string.

   ( ) n) n ∗