Informatics 2A 2018–19
Tutorial Sheet 3 (Week 5)

Mary Cryan

This week's exercises cover the basics of context-free grammars, pushdown automata, LL(1) grammars and predictive parsing (Lectures 9–12).

Question 4 (on constructing parse tables) represents perhaps the most technically demanding topic of the entire course, and needs the material from Lecture 12 (Friday 12th Oct). You should make sure you are comfortable with the ideas in Question 3 before attempting Question 4.

Throughout this sheet, we use n to stand for some lexical class of *numeric literals*, numeric literals being items like 5 or 23. The precise definition of this class is not relevant to these exercises.

1. In this question, ideas from the course turn and bite their own tail. We consider a context-free grammar for the (mathematical) language of regular expressions!

   Fix some alphabet Σ. The terminals of our grammar will be the symbols

   $$\mathsf{sym} \qquad \boldsymbol{\emptyset} \qquad \boldsymbol{\epsilon} \qquad + \qquad * \qquad (\qquad)$$

   where sym denotes the lexical class consisting of symbols from Σ. (Note that $\boldsymbol{\epsilon}$ is an actual symbol here, in contrast to ϵ which denotes the empty string.) There are two nonterminals: RegExp (the start symbol) and Atom. The productions are as follows:

   $$\begin{array}{rcl} \mathsf{RegExp} & \to & \mathsf{Atom} \mid \mathsf{RegExp} + \mathsf{RegExp} \mid \mathsf{RegExp}\ \mathsf{RegExp} \\ & & \mid \mathsf{RegExp} * \mid (\mathsf{RegExp}) \\ \mathsf{Atom} & \to & \mathsf{sym} \mid \boldsymbol{\emptyset} \mid \boldsymbol{\epsilon} \end{array}$$

   Note that this grammar represents (as is often done in practice) the concatenation of regular languages by juxtaposition of regular expressions, instead of using an explicit infix '.' operation.

   In Lecture 5, we gave the following mathematical definition of the language $\mathcal{L}(e)$ associated with a regular expression e.

   - $\mathcal{L}(\boldsymbol{\emptyset}) = \emptyset, \quad \mathcal{L}(\boldsymbol{\epsilon}) = \{\epsilon\}, \quad \mathcal{L}(a) = \{a\} \quad (a \in \Sigma)$
   - $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$
   - $\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha) \, . \, \mathcal{L}(\beta)$
   - $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$

   Strictly speaking, what $\mathcal{L}$ operates on here is not just a regular expression e in textual form, but a particular *syntax tree* for e: we need to know how e is parsed in order to make sense of $\mathcal{L}(e)$.

   (a) Give an example of a 'harmless' ambiguity arising from the above grammar; i.e., an ambiguous string for which alternative syntax trees all give rise to the same regular language. Draw two different syntax trees for this string.

(b) Give an example of a 'harmful' ambiguity; i.e., a string for which different syntax trees describe different regular languages. Draw two syntax trees for the string that do indeed describe different regular languages.

2. An NPDA $(Q, \Sigma, \Gamma, s, \Delta)$ (as defined in the Lecture 9 slides) is said to be *real-time deterministic* if it has no ϵ-transitions, and for every $q \in Q$, $a \in \Sigma$, $x \in \Gamma$ there is *at most one* pair (q', β) with $((q, a, x), (q', \beta)) \in \Delta$. [1] When working with deterministic PDAs, it is usual to assume the input alphabet Σ includes a special end-of-input symbol \$ which is required to occur at the end of every input string (and nowhere else).

   Design real-time deterministic PDAs that accept the following languages, using acceptance by empty stack.

   (a) The language $\{a^n b^n \mid n \geq 0\}$. (Hint: have two control states for the NPDA (often in class examples, we've had just a single control state).)

   (b) The set of well-matched bracket sequences involving two kinds of brackets: $(\ldots)$ and $[\ldots]$. For example, [([ ] ( ))] is a well-matched sequence, while ([ )] is not.

3. Consider the following grammar for generating arithmetic expressions such as $(\,\mathsf{n} * \mathsf{n} * \mathsf{n})$.

$$
\begin{array}{rl}
\text{Terminals:} & (,\ ),\ *,\ \mathsf{n} \\
\text{Nonterminals:} & \mathsf{Exp},\ \mathsf{Ops} \\
\text{Productions:} & \mathsf{Exp} \ \rightarrow\ \mathsf{n}\ \mathsf{Ops} \ \mid\ (\,\mathsf{Exp}\,) \\
 & \mathsf{Ops} \ \rightarrow\ \epsilon \ \mid\ *\ \mathsf{n}\ \mathsf{Ops} \\
\text{Start symbol:} & \mathsf{Exp}
\end{array}
$$

   Convince yourself that this is in fact an LL(1) grammar, and that its parse table is

	(	)	$*$	n	\$
Exp	(Exp)			n Ops	
Ops		ϵ	$*$ n Ops		ϵ

   (Here, for example, the top left entry (Exp) stands for the production $\mathsf{Exp} \rightarrow (\,\mathsf{Exp}\,)$.)

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

$$(\,\mathsf{n} * \mathsf{n}\,)$$

   At each step, show the operation applied, the input string remaining, and the stack state, as in the example on Lecture 11, Slide 9.

[1] The reason for the adjective "real-time" is that the standard notion of *deterministic PDA* is more general by allowing ϵ-transitions in certain circumstances, see, e.g., Chapter F of Kozen.

(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 reasonably issue to the author of the input string.

$$(\,) \qquad \mathsf{n} \,) \qquad \mathsf{n} \, *$$

(c) The grammar in this question is unnaturally restrictive. For example, it does not admit the string $(\,\mathsf{n} * \mathsf{n}\,) * \mathsf{n}$. Design an LL(1) grammar that recognises the full language of arithmetic expressions such as $(\,\mathsf{n} * \mathsf{n}\,) * (\mathsf{n} * \mathsf{n} * (\mathsf{n} * \mathsf{n}))$; i.e., the language of arithmetic expressions involving just $*$ and n, but allowing unrestricted (well-matched) bracketing.

4. Consider the following grammar for generating boolean conditions such as $\mathsf{n} + - \mathsf{n} * \mathsf{n} == \mathsf{n}$.

Terminals:	n, $+$, $*$, $-$, $==$
Nonterminals:	Cond, Exp, TimesExp, OptMinus, TimesOps, PlusOps
Productions:	Cond $\to$ Exp $==$ Exp
	Exp $\to$ TimesExp PlusOps
	TimesExp $\to$ OptMinus n TimesOps
	OptMinus $\to$ ϵ | $-$
	TimesOps $\to$ ϵ | $*$ n TimesOps
	PlusOps $\to$ ϵ | $+$ TimesExp PlusOps
Start symbol:	Cond

(a) Identify the set of nonterminals from which the empty string can be derived.

(b) Calculate the *First* sets for each of the nonterminals in the grammar.

(c) Calculate the *Follow* sets for each of the nonterminals in the grammar.

(d) Using this information, try to build a *parse table* for the grammar. Is the grammar LL(1) or not?