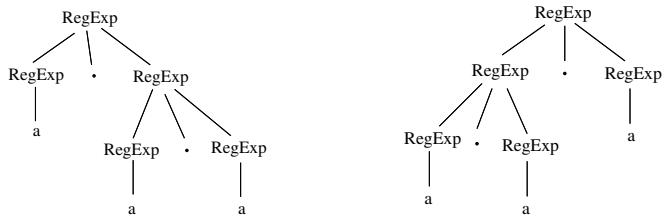
Informatics 2A: Tutorial Sheet 3 - SOLUTIONS

Mary Cryan

1. Suppose $a \in \Sigma$.

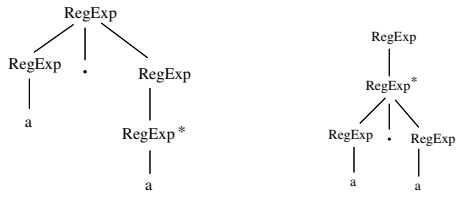
   (a) The string aaa can be derived in two ways with the following syntax trees.



   (note I have written a instead of `sym` in this case, taking a small bit of notational leniency).

   This ambiguity is harmless since both trees define the language $\{aaa\}$.

   (b) The string $aa*$ can also be derived in two ways, as shown below:



   (again I have taken the notational leniency of directly writing a). This ambiguity is harmful since one parse tree defines the regular language $\{a^n \mid n \geq 1\}$ and the other defines $\{a^{2n} \mid n \geq 0\}$. Note that the first parsing is the one that respects the usual precedence conventions.

2. (a) The idea is that we jump to the second state once the b's start coming. Formally, the control states are $Q = \{q_a, q_b\}$, the stack alphabet is $\Gamma = \{\bot\}$, and we include the following transitions:

$$
\begin{array}{lcl}
q_a & \xrightarrow{\$,\bot:\epsilon} & q_a \\
q_a & \xrightarrow{a,\bot:\bot\bot} & q_a \\
q_a & \xrightarrow{b,\bot:\epsilon} & q_b \\
q_b & \xrightarrow{b,\bot:\epsilon} & q_b \\
q_b & \xrightarrow{\$,\bot:\epsilon} & q_b
\end{array}
$$

(N.B., the $ symbol can only ever occur at the end of the input string.)

Strictly speaking, we should also add a garbage state as the destination for all remaining transitions. The self transitions on the garbage state need to ensure that the stack never empties at the end-of-input marker $. This can be implemented by including a self-transition $\$, \bot : \bot$ on the garbage state.

(b) Let $\Gamma = \{(, [, \bot\}$. Consider a single state with the following self-transitions.

$$
\begin{array}{ll}
(, x : (x & \text{for each } x \in \Gamma \\
[, x : [x & \text{for each } x \in \Gamma \\
), (: \epsilon & \\
], [: \epsilon & \\
\$, \bot : \epsilon &
\end{array}
$$

Again, strictly speaking, one should add a garbage state as destination for all remaining transitions.

3. (a)

Operation	Input remaining	Stack state
	(n * n)$	Exp
Lookup (,Exp	(n * n)$	(Exp)
Match (	n * n)$	Exp)
Lookup n, Exp	n * n)$	n Ops)
Match n	* n)$	Ops)
Lookup *, Ops	* n)$	* n Ops)
Match *	n)$	n Ops)
Match n	)$	Ops)
Lookup), Ops	)$	)
Match)	$	STACK EMPTIES AT END OF STRING: SUCCESS!

(b)
- For `()`, the parser will encounter a blank table entry at `), Exp`.
  Message: "`)` Found where expression expected."
- For `n)`, the stack will empty before end of input is reached.
  Message: "`)` Found after end of expression."
- For `n*`, the end of input will be reached with `n Ops` still on the stack, and the parser gets stuck since the top of the stack is a terminal `n` no different from `$`.
  Message: "End of input found where numeric literal expected."

(c)

$$
\begin{array}{rcl}
\mathsf{Exp} & \rightarrow & \mathsf{ExpA\ Ops} \\
\mathsf{Ops} & \rightarrow & \epsilon \mid * \mathsf{\ ExpA\ Ops} \\
\mathsf{ExpA} & \rightarrow & \mathsf{n} \mid \mathsf{(\ Exp\ )}
\end{array}
$$

(Other solutions are possible.)

4. (a) E = {OptMinus, TimesOps, PlusOps}

   (b) $First$(OptMinus) = $\{-, \epsilon\}$
   $First$(TimesOps) = $\{*, \epsilon\}$
   $First$(PlusOps) = $\{+, \epsilon\}$
   $First$(Exp) = $First$(Cond) = $First$(TimesExp) = {−,n}

   (c) $Follow$(Cond) = {$}
   $Follow$(Exp) = {$, ==}
   $Follow$(PlusOps) = {$, ==}
   $Follow$(TimesExp) = {$, ==, +}
   $Follow$(TimesOps) = {$, ==, +}
   $Follow$(OptMinus) = {n}

   (d) The grammar is indeed LL(1)!
   Parse table as follows.

	n	+	*	-	==	$
Cond	Exp==Exp			Exp==Exp		
Exp	TimesExp PlusOps			TimesExp PlusOps		
TimesExp	OptMinus n TimesOps			OptMinus n TimesOps		
OptMinus	ϵ			-		
TimesOps		ϵ	* n TimesOps		ϵ	ϵ
PlusOps		+ TimesExp PlusOps			ϵ	ϵ