Chapter 4 Top-Down Parsing

Ex 4.8

(a)
lexp $\rightarrow$ atom | list
atom $\rightarrow$ number | identifier
list $\rightarrow$ (lex-seq)
lex-seq $\rightarrow$ lexp lex-sep'
lex-seq’ $\rightarrow$ lexp lex-seq’ | ε

(b)
First(lexp) = { number, identifier, (}
First(atom) = { number, identifier }
First(list) = { (}
First(lex-seq) = { number, identifier, (}
FIrst(lex-seq’) = { number, identifier, (, ε }

Follow(lexp) = { \$, number, identifier, (,) }
Follow(lex-seq’) = {) }
Follow(atom) = { \$, number, identifier, (,) }
Follow(list) = { \$, number, identifier, (,) }
Follow(lex-seq) = {) }

(c)
First(atom) $\bigcap$ First(list) = $\varnothing$
First(lex-seq’) $\bigcap$ Follow(lex-seq’) = $\varnothing$
So the grammer is LL(1)

(d)

M[N,T]	number	identifier	(	)	$
lexp	lexp → atom	lexp → atom	lexp → list		
atom	atom → number	atom → identifier			
list			list → (lexp-seq)		
lex-seq	lex-seq → lexp lex-seq’	lex-seq → lexp lex-seq’	lex-seq → lexp lex-seq’		
lex-seq’	lex-seq’ → lexp lex-seq’	lex-seq’ → lexp lex-seq’	lex-seq’ → lexp lex-seq’	lex-seq’ → ε	

(e)

Analysis stack	Input	Action
$ lexp	(a (b (2)) (c)) $	lexp → list
$ list	(a (b (2)) (c)) $	list → (lexp-seq)
$) lex-seq (	(a (b (2)) (c)) $	match
$) lex-seq	a (b (2)) (c)) $	lex-seq → lexp lex-seq’
$) lex-seq’ lexp	a (b (2)) (c)) $	lexp → atom
$) lex-seq’ atom	a (b (2)) (c)) $	atom → identifier
$) lex-seq’ identifier	a (b (2)) (c)) $	match
$) lex-seq’	(b (2)) (c)) $	lex-seq’ → lexp lex-seq’
$) lex-seq’ lexp	(b (2)) (c)) $	lexp → list
$) lex-seq’ list	(b (2)) (c)) $	list → (lexp-seq)
$) lex-seq’) lex-seq (	(b (2)) (c)) $	match
$) lex-seq’) lex-seq	b (2)) (c)) $	lex-seq → lexp lex-seq’
$) lex-seq’) lex-seq’ lexp	b (2)) (c)) $	lexp → atom
$) lex-seq’) lex-seq’ atom	b (2)) (c)) $	atom → identifier

Analysis stack	Input	Action
$) lex-seq’) lex-seq’ identifier	b (2)) (c)) $	match
$) lex-seq’) lex-seq’	(2)) (c)) $	lex-seq’ → lexp lex-seq’
$) lex-seq’) lex-seq’ lexp	(2)) (c)) $	lexp → list
$) lex-seq’) lex-seq’) lex-seq (	(2)) (c)) $	match
$) lex-seq’) lex-seq’) lex-seq	2)) (c)) $	lex-seq → lexp lex-seq’
$) lex-seq’) lex-seq’) lex-seq’ lexp	2)) (c)) $	lexp → atom
$) lex-seq’) lex-seq’) lex-seq’ atom	2)) (c)) $	atom → number
$) lex-seq’) lex-seq’) lex-seq’ number	2)) (c)) $	match
$) lex-seq’) lex-seq’) lex-seq’	)) (c)) $	lex-seq’ → ε
$) lex-seq’) lex-seq’)	)) (c)) $	mtach
$) lex-seq’) lex-seq’	) (c)) $	lex-seq’ → ε
$) lex-seq’)	) (c)) $	match
$) lex-seq’	(c)) $	lex-seq’ → lexp lex-seq’
$) lex-seq’ lexp	(c)) $	lexp → list
$) lex-seq’ list	(c)) $	list → (lexp-seq)
$) lex-seq’) lex-seq (	(c)) $	match
$) lex-seq’) lex-seq	c)) $	lex-seq → lexp lex-seq’
$) lex-seq’) lex-seq’ lexp	c)) $	lexp → atom

Analysis stack	Input	Action
\$) lex-seq’) lex-seq’ atom	c)) \$	atom $\rightarrow$ identifier
\$) lex-seq’) lex-seq’ identifier	c)) \$	mtcah
\$) lex-seq’) lex-seq’	)) \$	lex-seq’ $\rightarrow \varepsilon$
\$) lex-seq’)	)) \$	match
\$) lex-seq’	) \$	lex-seq’ $\rightarrow \varepsilon$
\$)	) \$	match
\$	\$	mtach

Ex 4.12

a.
No, it can’t. Because the parsing table of a LL(1) grammar has unique entrances.

b.
No, an ambiguous grammar can’t be an LL(1) grammar.

c.
A non-ambiguous grammar may be or not be an LL(1) grammar. LL(1) grammar is just a kind of non-ambiguous grammars.