nHomework 4.8, 4.12

4.8 Consider the grammar

lexp $\rightarrow$ *atom* | *list*

atom $\rightarrow$ ***number* | *identifier***

list $\rightarrow$ *(lexp-seq)*

lexp-seq $\rightarrow$ *lexp-seq lexp* | *lexp*

a. Remove the left recursion

b. Construct First and Follow sets for the nonterminals of the resulting grammar.

c. Show that the resulting grammar is LL(1)

d. Construct the LL(1) parsing table for the resulting grammar .

e. Show the actions of the corresponding LL(1) parser, given the input string **(a (b (2)) (c))**.

a.

lexp $\rightarrow$ *atom* | *list*

atom $\rightarrow$ ***number* | *identifier***

list $\rightarrow$ *(lexp-seq)*

lexp-seq $\rightarrow$ *lexp lexp-seq'*

lexp-seq' $\rightarrow$ *lexp lexp-seq'* | ε

b.

	First	Follow
lexp	{ (, ***number***, ***identifier*** }	{ $, (,), ***number***, ***identifier*** }
atom	{ ***number***, ***identifier*** }	{ $, (,), ***number***, ***identifier*** }
list	{ (}	{ $, (,), ***number***, ***identifier*** }
lexp-seq	{ (, ***number***, ***identifier*** }	{) }
lexp-seq'	{ (, ***number***, ***identifier***, ε }	{) }

c.

According to the theorem:

First(*atom*) $\cap$ First(*list*) = ϕ

First(*lexp lexp-seq'*) $\cap$ First(ε) = ϕ

First(*lexp-seq'*) $\cap$ Follow(*lexp-seq'*) = ϕ

d.

	number	***identifier***	(	)	ε
lexp	*lexp* $\rightarrow$ *atom*	*lexp* $\rightarrow$ *atom*	*lexp* $\rightarrow$ *list*		
atom	*atom* $\rightarrow$ ***number***	*atom* $\rightarrow$ ***identifier***			
list			*list* $\rightarrow$ *(lexp-seq)*		
lexp-seq	*lexp-seq* $\rightarrow$ *lexp lexp-seq'*	*lexp-seq* $\rightarrow$ *lexp lexp-seq'*	*lexp-seq* $\rightarrow$ *lexp lexp-seq'*		
lexp-seq'	*lexp-seq'* $\rightarrow$ *lexp*	*lexp-seq'* $\rightarrow$ *lexp*	*lexp-seq'* $\rightarrow$	*lexp-seq'* $\rightarrow$	

	lexp-seq’	*lexp-seq’*	*lexp lexp-seq’*	ε	

e.

Parsing stack	Input	Action
$ *lexp*	(a (b (2)) (c))$	*lexp* $\rightarrow$ *list*
$ *list*	(a (b (2)) (c)) $	*list* $\rightarrow$ *(lexp-seq)*
$ *) lexp-seq (*	(a (b (2)) (c)) $	match
$ *) lexp-seq*	a (b (2)) (c)) $	*lexp-seq* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’ lexp*	a (b (2)) (c)) $	*lexp* $\rightarrow$ *atom*
$ *) lexp-seq’ atom*	a (b (2)) (c)) $	*atom* $\rightarrow$ ***identifier***
$ *) lexp-seq’* ***identifier***	a (b (2)) (c)) $	match
$ *) lexp-seq’*	(b (2)) (c)) $	*lexp-seq’* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’ lexp*	(b (2)) (c)) $	*lexp* $\rightarrow$ *list*
$ *) lexp-seq’ list*	(b (2)) (c)) $	*list* $\rightarrow$ *(lexp-seq)*
$ *) lexp-seq’) lexp-seq (*	(b (2)) (c)) $	match
$ *) lexp-seq’) lexp-seq*	b (2)) (c)) $	*lexp-seq* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’) lexp-seq’ lexp*	b (2)) (c)) $	*lexp* $\rightarrow$ *atom*
$ *) lexp-seq’) lexp-seq’ atom*	b (2)) (c)) $	*atom* $\rightarrow$ ***identifier***
$ *) lexp-seq’) lexp-seq’* ***identifier***	b (2)) (c)) $	match
$ *) lexp-seq’) lexp-seq’*	(2)) (c)) $	*lexp-seq’* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’) lexp-seq’ lexp*	(2)) (c)) $	*lexp* $\rightarrow$ *list*
$ *) lexp-seq’) lexp-seq’ list*	(2)) (c)) $	*list* $\rightarrow$ *(lexp-seq)*
$ *) lexp-seq’) lexp-seq’) lexp-seq (*	(2)) (c)) $	macth
$ *) lexp-seq’) lexp-seq’) lexp-seq*	2)) (c)) $	*lexp-seq* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’) lexp-seq’) lexp-seq’ lexp*	2)) (c)) $	*lexp* $\rightarrow$ *atom*
$ *) lexp-seq’) lexp-seq’) lexp-seq’ atom*	2)) (c)) $	*atom* $\rightarrow$ ***number***
$ *) lexp-seq’) lexp-seq’) lexp-seq’* ***number***	2)) (c)) $	match
$ *) lexp-seq’) lexp-seq’) lexp-seq’*	)) (c)) $	*lexp-seq’* $\rightarrow$ ε
$ *) lexp-seq’) lexp-seq’)*	)) (c)) $	match
$ *) lexp-seq’) lexp-seq’*	) (c)) $	*lexp-seq’* $\rightarrow$ ε
$ *) lexp-seq’)*	) (c)) $	match
$ *) lexp-seq’*	(c)) $	*lexp-seq’* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’ lexp*	(c)) $	*lexp* $\rightarrow$ *list*
$ *) lexp-seq’ list*	(c)) $	*list* $\rightarrow$ *(lexp-seq)*
$ *) lexp-seq’) lexp-seq (*	(c)) $	match
$ *) lexp-seq’) lexp-seq*	c)) $	*lexp-seq* $\rightarrow$ *lexp lexp-seq’*
$ *) lexp-seq’) lexp-seq’ lexp*	c)) $	*lexp* $\rightarrow$ *atom*
$ *) lexp-seq’) lexp-seq’ atom*	c)) $	*atom* $\rightarrow$ ***identifier***
$ *) lexp-seq’) lexp-seq’* ***identifier***	c)) $	match
$ *) lexp-seq’) lexp-seq’*	)) $	*lexp-seq’* $\rightarrow$ ε
$ *) lexp-seq’)*	)) $	match
$ *) lexp-seq’*	) $	*lexp-seq’* $\rightarrow$ ε
$ *)*	) $	match

$	$	accept

4.12

a. Can an LL(1) grammar be ambiguous? Why or why not ?

b. Can an ambiguous grammar be LL(1) ? Why or why not?

c. Must an unambiguous grammar be LL(1) ? Why or why not ?

a.

No. Since the leftmost derivation constructed is unique.

b.

No. The reason is the same as the last question. The grammar should be unambiguous.

c.

No. It is easy to construct an unambiguous grammar that is not LL(1), which may be transformed into LL(1).