练习 4.2.1：使得文法的预测分析产生回溯的原因是什么？仅使用 FIRST 集合可以避免回溯吗？为什么？

即使当非终结符用某个产生式匹配成功，但是这种成功可能只是暂时的，因为没有足够的信息来唯一地确定可能的产生式，所以分析过程就会产生回溯。
不可以。例如对于产生式 A=>α|β, FIRST(α) 与 FIRST(β) 交集为空集，但ϵ是其中某个 FIRST 集合的元素，不失一般性，假设$\epsilon \in$ FIRST(α)，想要避免回溯，则还需要考虑 FOLLOW(A)与 FIRST(β)的情况。

练习 4.2.2：考虑文法：

lexp→atom | list
atom→number | identifier
list→(lexp-seq)
lexp-seq→lexp-seq lexp | lexp

1）消除左递归
2）求得该文法的 FIRST 集合和 FOLLOW 集合
3）说明所得的文法是 LL(1)文法
4）为所得的文法构造 LL(1)分析表
5）对输入串(a (b (2)) (c))给出相应得 LL(1)分析程序的动作

1）

$$
\begin{aligned}
\text{lexp} &\rightarrow \text{atom} \mid \text{list} \\
\text{atom} &\rightarrow \text{number} \mid \text{identifiler} \\
\text{list} &\rightarrow (\ \text{lexp-seq}\ ) \\
\text{lexp-seq} &\rightarrow \text{lexp lexp-seq'} \\
\text{lexp-seq'} &\rightarrow \text{lexp lexp-seq'} \mid \varepsilon
\end{aligned}
$$

2)

	FIRST	FOLLOW
lexp	number, identifier, (	$,), number, identifier, (
atom	number, identifier	$,), number, identifier, (
list	(	$,), number, identifier, (
lexp-seq	number, identifier, (	)
lexp-seq'	ϵ, number, identifier, (	)

3)
可以根据 LL(1) 文法的定义来证明。因为对于：
- lexp 为左部的产生式，有 FIRST(atom) 和 FIRST(list)不相交；
- atom 为左部的产生式，FIRST(number) 和 FIRST(identifier)不相交；
- lexp-seq’ 为左部的产生式，FIRST(lexp lexpseq') 和 FIRST(ϵ)不相交，且 FIRST(lexp lexp-seq') 和 FOLLOW(lexp-seq

所以该文法是 LL(1) 文法

4.
4)

	number	identifier	(	)	$
lexp	lexp→atom	lexp→atom	lexp→list		
atom	atom→number	atom→identifier			
list			list→(lexp-seq)		
lexp-seq	lexp-seq→lexp lexp-seq'	lexp-seq→lexp lexp-seq'	lexp-seq→lexp lexp-seq'		
lexp-seq'	lexp-seq'→lexp lexp-seq'	lexp-seq'→lexp lexp-seq'	lexp-seq'→ lexp lexp-seq'	lexp-seq'→ ϵ	

5)

Stack	Input	Rule
$lexp	(a(b(2))(c))$	
$list	(a(b(2))(c))$	lexp->list
$)lexp-seq(	(a(b(2))(c))$	list->(lexp-seq)
$)lexp-seq	a(b(2))(c))$	match
$)lexp-seq'lexp	a(b(2))(c))$	lexp-seq->lexp lexp-seq'
$)lexp-seq'atom	a(b(2))(c))$	lexp->atom
$)lexp-seq'identifier	a(b(2))(c))$	atom-> identifier
$)lexp-seq'	(b(2))(c))$	match
$)lexp-seq'lexp	(b(2))(c))$	lexp-seq'->lexp lexp-seq'
$)lexp-seq'list	(b(2))(c))$	lexp->list
$)lexp-seq')lexp-seq(	(b(2))(c))$	list->(lexp-seq)
$)lexp-seq')lexp-seq	b(2))(c))$	match
$)lexp-seq')lexp-seq'lexp	b(2))(c))$	lexp-seq->lexp lexp-seq'
$)lexp-seq')lexp-seq'atom	b(2))(c))$	lexp->atom
$)lexp-seq')lexp-seq'identifier	b(2))(c))$	atom->identifier
$)lexp-seq')lexp-seq'	(2))(c))$	match
$)lexp-seq')lexp-seq'lexp	(2))(c))$	lexp-seq'->lexp lexp-seq'
$)lexp-seq')lexp-seq'list	(2))(c))$	lexp->list
$)lexp-seq')lexp-seq')lexp-seq(	(2))(c))$	list->(lexp-seq)
$)lexp-seq')lexp-seq')lexp-seq	2))(c))$	match
$)lexp-seq')lexp-seq')lexp-seq'lexp	2))(c))$	lexp-seq->lexp lexp-seq'
$)lexp-seq')lexp-seq')lexp-seq'atom	2))(c))$	lexp->atom
$)lexp-seq')lexp-seq')lexp-seq' number	2))(c))$	atom->number
$)lexp-seq')lexp-seq')lexp-seq'	))(c))$	match

$) lexp-seq') lexp-seq')	))(c))$	lexp-seq' ->**ε**
$) lexp-seq') lexp-seq'	)(c))$	match
$) lexp-seq')	)(c))$	lexp-seq' ->**ε**
$) lexp-seq'	(c))$	match
$) lexp-seq' lexp	(c))$	lexp-seq'->lexp lexp-seq'
$) lexp-seq' list	(c))$	lexp -> list
$) lexp-seq') lexp-seq (	(c))$	list -> (lexp-seq)
$) lexp-seq') lexp-seq	c))$	match
$) lexp-seq') lexp-seq' lexp	c))$	lexp-seq->lexp lexp-seq'
$) lexp-seq') lexp-seq' atom	c))$	lexp -> atom
$) lexp-seq') lexp-seq' identifier	c))$	atom -> identifier
$) lexp-seq') lexp-seq'	))$	match
$) lexp-seq')	))$	lexp-seq' -> **ε**
$) lexp-seq'	)$	match
$)	)$	lexp-seq' ->**ε**
$	$	accept