① ⓐ

X	FIRST(X)	FOLLOW(X)
S	n , (	∅
E	n , (	n , (,) , $

1. $S \to E\$$
2. $E \to n$
3. $E \to (\lambda n.E)$
4. $E \to (E\,E)$

ⓑ (∈ PREDICT(3) ∩ PREDICT(4) ⇒ not LL(1)

G_1': common prefix elimination:

1. $S \to E\$$
2. $E \to n$
3. $E \to (F$
4. $F \to \lambda n.E)$
5. $F \to E\,E)$

G_1' is LL(1)

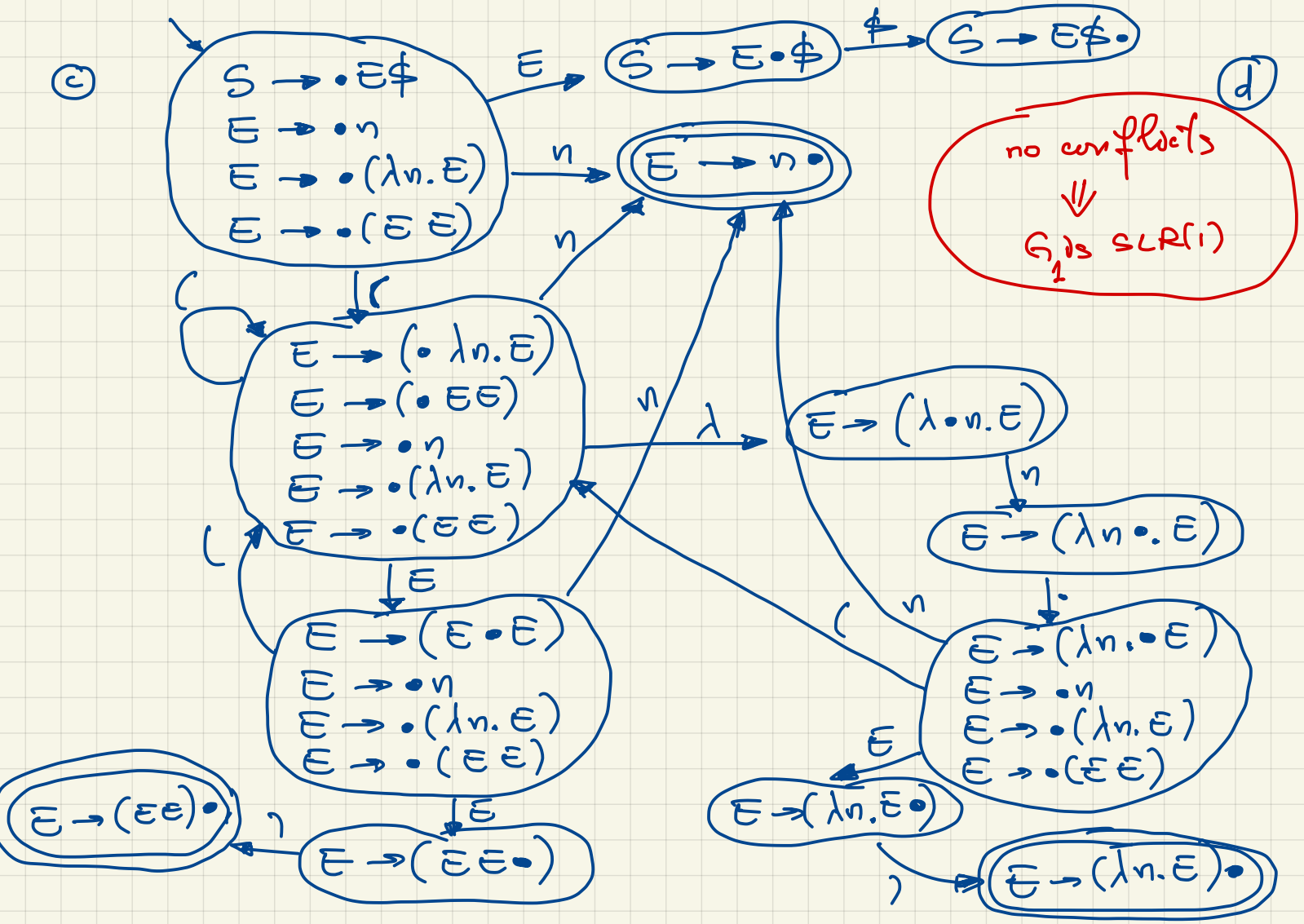
PREDICT

n, (

no intersection { n, (

no intersection { λ, n, (

ⓒ



ⓓ

no conflicts

⇓

G_1' is SLR(1)

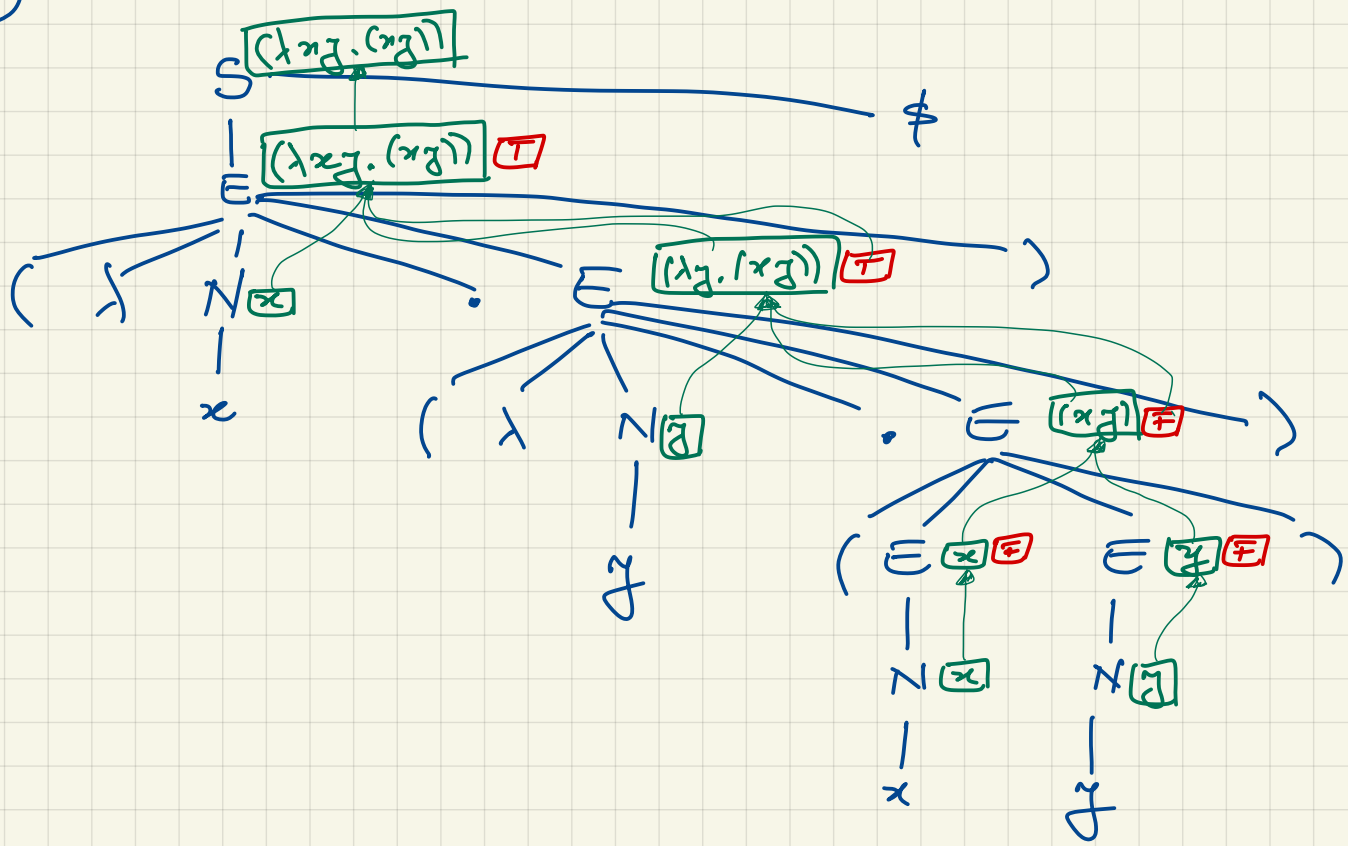
isλ = T for abstractions, F otherwise
str = the string

(2) (a)

$S \to E\,\$$	$S.str = E.str$
$E \to N$	$E.str = N.str$
$E_1 \to (\lambda N.E_2)$	$E_1.str = \text{if}(E_2.isλ)\ \text{then}\ "(\lambda" + N.str + E_2.str[2..len-2] + ")"$ $\text{else}\ "(\lambda" + N.str + "." + E_2.str + ")"$ $E_1.isλ = T$
$E_1 \to (E_2 E_3)$	$E_1.str = "(" + E_2.str + E_3.str + ")"$ $E_1.isλ = F$
$N \to a \mid b \mid \cdots \mid z$	$N.str = 'a' \mid 'b' \mid \cdots \mid 'z'$

(b) $(\lambda xyz.((xy)z))\,\$$

(c)



(3) (a) The first function does exponential work because it uses an exponential number of recursive calls. The second function runs in linear time. The last function runs in logarithmic time, necessary for the exponentiation. Therefore, the first function is the slowest.

(b) There is no implication on the competition between recursive and iterative speed. The first function is the slowest not because it is recursive, but because it recomputes the same values many times, resulting in exponential amount of work. The second function is iterative and runs in linear time. The third is neither iterative, nor recursive, therefore its speed has no bearing on the issue.

④ (a) $(\lambda x.x)(\lambda x.x)$

$\Rightarrow_\beta \lambda x.x$

(b) $(\lambda b.a(\lambda b.bb)a)(\lambda a.((\lambda b.b)a))$

$\Rightarrow_\beta (\lambda b.a(\lambda b.bb)a)(\lambda a.a)$

$\Rightarrow_\beta a(\lambda b.bb)a$

(c) $(\lambda x.xx)(\lambda x.xxx)$

$\Rightarrow_\beta (\lambda x.xxx)(\lambda x.xxx)$

$\Rightarrow_\beta (\lambda x.xxx)(\lambda x.xxx)(\lambda x.xxx)$

$\Rightarrow_\beta (\lambda x.xxx)(\lambda x.xxx)(\lambda x.xxx)(\lambda x.xxx)$

$\Rightarrow_\beta \cdots$ there is no normal form

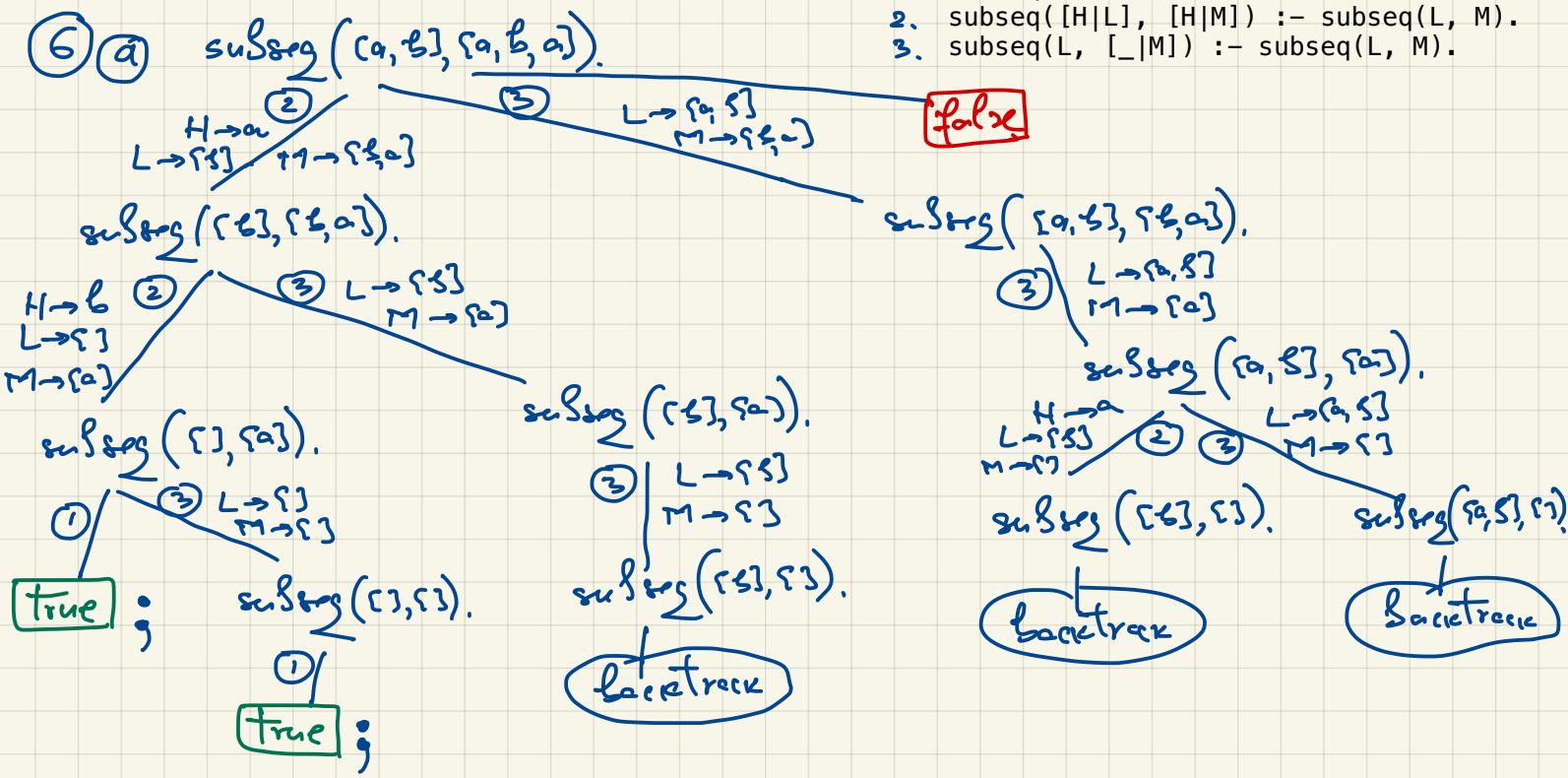
5

```
(define member-twice
  (lambda (x L)
    (letrec ((member-once
              (lambda (x L)
                (if (null? L) #f
                    (if (equal? x (car L)) #t
                        (member-once x (cdr L)))))))
      (if (or (null? L) (null? (cdr L)))
          #f
          (if (equal? x (car L))
              (member-once x (cdr L))
              (member-twice x (cdr L)))))))

(member-twice 'a '()) ; => #f
(member-twice 'a '(a)) ; => #f
(member-twice 'a '(a a)) ; => #t
(member-twice 'a '(a b b c)) ; => #f
(member-twice 'a '(b a c a a)) ; => #t
```

```
1. subseq([], _).
2. subseq([H|L], [H|M]) :- subseq(L, M).
3. subseq(L, [_|M]) :- subseq(L, M).
```

6 a)




```
1. subseq1([], _).
2. subseq1([H|L], [H|M]) :- subseq1(L, M).
3. subseq1([H|L], [X|M]) :- not(H = X), subseq1([H|L], M).
```

6 b)