Distribute the following questions across the members of your group. You will share your solutions (and most importantly the *method* of your solutions) during the next lecture period. Divide up the questions so that **each** question has at least two solutions from different group members.

Questions for Review

1. Refactor the following grammars by consolidating the common prefixes until they are LL(1).

(a)

$$
\begin{array}{rcl}
S & \rightarrow & a\ b\ c\ d\ e\ \$ \\
 & | & a\ b\ c\ q\ y\ z\ \$ \\
 & | & a\ b\ c\ q\ r\ s\ \$ \\
 & | & T\ U\ \$ \\
T & \rightarrow & x\ a\ b \\
 & | & \lambda \\
U & \rightarrow & G\ z \\
G & \rightarrow & d \\
 & | & q
\end{array}
$$

(b)

$$
\begin{array}{rcl}
START & \rightarrow & A\ x\ B\ \$ \\
 & | & C\ \$ \\
 & | & A\ q\ C\ r\ \$ \\
A & \rightarrow & x\ y\ B \\
B & \rightarrow & g\ h\ m \\
 & | & g\ k\ n \\
 & | & g\ h\ q \\
C & \rightarrow & d\ b\ f \\
 & | & \lambda
\end{array}
$$

(c)

$$
\begin{array}{rcl}
START & \rightarrow & S\ \$ \\
S & \rightarrow & a\ S\ e \\
 & | & B \\
B & \rightarrow & b\ a\ e\ B\ e \\
 & | & a\ e\ C \\
 & | & b\ a\ e\ B\ g \\
C & \rightarrow & c\ c\ C\ e \\
 & | & c\ c\ B\ d
\end{array}
$$

Solution: question 1 part a

Original

$$
\begin{array}{rcl}
S & \rightarrow & a\ b\ c\ d\ e\ \$ \\
 & | & a\ b\ c\ q\ y\ z\ \$ \\
 & | & a\ b\ c\ q\ r\ s\ \$ \\
 & | & T\ U\ \$ \\
T & \rightarrow & x\ a\ b \\
 & | & \lambda \\
U & \rightarrow & G\ z \\
G & \rightarrow & d \\
 & | & q
\end{array}
$$

Add symbol H to resolve conflicts for S

$$
\begin{array}{rcl}
S & \rightarrow & a\ b\ c\ H\ \$ \\
 & | & T\ U\ \$ \\
H & \rightarrow & d\ e \\
 & | & q\ y\ z \\
 & | & q\ r\ s \\
T & \rightarrow & x\ a\ b \\
 & | & \lambda \\
U & \rightarrow & G\ z \\
G & \rightarrow & d \\
 & | & q
\end{array}
$$

$\Rightarrow$

Add symbol J to resolve conflicts for H

$$
\begin{array}{rcl}
S & \rightarrow & a\ b\ c\ H\ \$ \\
 & | & T\ U\ \$ \\
H & \rightarrow & d\ e \\
 & | & q\ J \\
J & \rightarrow & y\ z \\
 & | & r\ s \\
T & \rightarrow & x\ a\ b \\
 & | & \lambda \\
U & \rightarrow & G\ z \\
G & \rightarrow & d \\
 & | & q
\end{array}
$$

Solution: question 1 part b

Original

$$
\begin{array}{rcl}
START & \rightarrow & A\ x\ B\ \$ \\
 & | & C\ \$ \\
 & | & A\ q\ C\ r\ \$ \\
A & \rightarrow & x\ y\ B \\
B & \rightarrow & g\ h\ m \\
 & | & g\ k\ n \\
 & | & g\ h\ q \\
C & \rightarrow & d\ b\ f \\
 & | & \lambda
\end{array}
$$

$\Rightarrow$

Add symbol H to resolve conflicts for $START$

$$
\begin{array}{rcl}
START & \rightarrow & A\ H\ \$ \\
 & | & C\ \$ \\
H & \rightarrow & x\ B \\
 & | & q\ C\ r \\
A & \rightarrow & x\ y\ B \\
B & \rightarrow & g\ h\ m \\
 & | & g\ k\ n \\
 & | & g\ h\ q \\
C & \rightarrow & d\ b\ f \\
 & | & \lambda
\end{array}
$$

$\Rightarrow$

Add symbol J to resolve conflicts for B

$$
\begin{array}{rcl}
START & \rightarrow & A\ H\ \$ \\
 & | & C\ \$ \\
H & \rightarrow & x\ B \\
 & | & q\ C\ r \\
A & \rightarrow & x\ y\ B \\
B & \rightarrow & g\ J \\
J & \rightarrow & h\ m \\
 & | & k\ n \\
 & | & h\ q \\
C & \rightarrow & d\ b\ f \\
 & | & \lambda
\end{array}
$$

$\Rightarrow$

Add symbol M to resolve conflicts for J

$$
\begin{array}{rcl}
START & \rightarrow & A\ H\ \$ \\
 & | & C\ \$ \\
H & \rightarrow & x\ B \\
 & | & q\ C\ r \\
A & \rightarrow & x\ y\ B \\
B & \rightarrow & g\ J \\
J & \rightarrow & h\ M \\
 & | & k\ n \\
M & \rightarrow & m \\
 & | & q \\
C & \rightarrow & d\ b\ f \\
 & | & \lambda
\end{array}
$$

Solution: question 1 part c
Simple rote approach for B and C conflicts fails...

#	Rules
1	$START \rightarrow S\ \$$
2	$S \rightarrow a\ S\ e$
3	$S \rightarrow B$
4	$B \rightarrow b\ a\ e\ B\ H$
5	$B \rightarrow a\ e\ C$
6	$H \rightarrow e$
7	$H \rightarrow g$
8	$C \rightarrow c\ c\ V$
9	$V \rightarrow C\ e$
10	$V \rightarrow B\ d$

S production rule conflicts for terminal a in $First(B)$...

	a	b	c	d	e	g	$
START	1	1					
S	⋆	3					
B	5	4					
H					6	7	
C			8				
V	10	10	9				

Original

$$\begin{aligned} START &\rightarrow S\ \$ \\ S &\rightarrow a\ S\ e \\ &\mid B \\ B &\rightarrow b\ a\ e\ B\ e \\ &\mid a\ e\ C \\ &\mid b\ a\ e\ B\ g \\ C &\rightarrow c\ c\ C\ e \\ &\mid c\ c\ B\ d \end{aligned}$$

$\Rightarrow$ Rewrite S

$$\begin{aligned} START &\rightarrow S\ \$ \\ S &\rightarrow a\ S\ e \\ &\mid b\ a\ e\ B\ e \\ &\mid a\ e\ C \\ &\mid b\ a\ e\ B\ g \\ B &\rightarrow b\ a\ e\ B\ e \\ &\mid a\ e\ C \\ &\mid b\ a\ e\ B\ g \\ C &\rightarrow c\ c\ C\ e \\ &\mid c\ c\ B\ d \end{aligned}$$

Add symbols T and U to resolve conflicts for S

$$\begin{aligned} START &\rightarrow S\ \$ \\ S &\rightarrow a\ T \\ &\mid b\ a\ e\ B\ U \\ T &\rightarrow S\ e \\ &\mid C \\ U &\rightarrow e \\ &\mid g \\ B &\rightarrow b\ a\ e\ B\ e \\ &\mid a\ e\ C \\ &\mid b\ a\ e\ B\ g \\ C &\rightarrow c\ c\ C\ e \\ &\mid c\ c\ B\ d \end{aligned}$$

$\Rightarrow$ Add symbols W and V to resolve conflicts for B and C

$$\begin{aligned} START &\rightarrow S\ \$ \\ S &\rightarrow a\ T \\ &\mid b\ a\ e\ B\ U \\ T &\rightarrow S\ e \\ &\mid C \\ U &\rightarrow e \\ &\mid g \\ B &\rightarrow b\ a\ e\ B\ W \\ &\mid a\ e\ C \\ W &\rightarrow e \\ &\mid g \\ C &\rightarrow c\ c\ V \\ V &\rightarrow C\ e \\ &\mid B\ d \end{aligned}$$

2. **("Double coverage" for this question can be one group member doing the coding, and another doing the testing.)** Incorporate the solution to question 3 of `lga-ll1-parsing.pdf` into your group's "grammar code"; test with `parser-test.tok.cfg` and input `parser-test.tok` — see also `show_llparse-parser-test.tok.pdf` .

3. Refactor these grammars' left-recursive rules to make the grammar LL(1) (some grammars may require common prefix refactoring as well).

(a)

$$
\begin{array}{rcl}
S & \rightarrow & Q\ R\ \$ \\
Q & \rightarrow & Q\ x\ Q\ y \\
 & | & \lambda \\
R & \rightarrow & R\ r\ s\ t\ x\ y \\
 & | & R\ r\ s\ t\ y\ y\ y \\
 & | & s\ t
\end{array}
$$

(b)

$$
\begin{array}{rcl}
S & \rightarrow & SUM\ \$ \\
SUM & \rightarrow & SUM\ plus\ PROD \\
 & | & PROD \\
PROD & \rightarrow & PROD\ mult\ POWER \\
 & | & POWER \\
POWER & \rightarrow & val\ exp\ POWER \\
 & | & val
\end{array}
$$

(c)

$$
\begin{array}{rcl}
S & \rightarrow & FUNCTIONS\ \$ \\
FUNCTIONS & \rightarrow & FUNCTIONS\ FUNCTION \\
 & | & FUNCTION \\
FUNCTION & \rightarrow & C \\
 & | & P \\
 & | & H \\
C & \rightarrow & type\ id\ oparen\ CPARAMS\ cparen \\
P & \rightarrow & def\ id\ oparen\ PPARAMS\ cparen \\
H & \rightarrow & id\ dblcln\ type\ HPARAMS \\
CPARAMS & \rightarrow & CPARAMS\ comma\ type\ id \\
 & | & \lambda \\
PPARAMS & \rightarrow & PPARAMS\ comma\ id \\
 & | & \lambda \\
HPARAMS & \rightarrow & HPARAMS\ rarrow\ type \\
 & | & type
\end{array}
$$

Solution: question 3 part a

$$
\begin{array}{lcl}
S & \rightarrow & Q\ R\ \$ \\
Q & \rightarrow & x\ Q\ y\ H \\
H & \rightarrow & x\ Q\ y\ H \\
 & | & \lambda \\
Q & \rightarrow & \lambda \\
R & \rightarrow & s\ t\ U\ V \\
V & \rightarrow & r\ s\ t\ U\ V \\
 & | & \lambda \\
U & \rightarrow & x\ y \\
 & | & y\ y\ y \\
 & | & \lambda
\end{array}
$$

```
S -> Q R $

# "Q" thought process
# gamma = x, beta = Q y fails because the
# rewrite (with new terminal H) would be
# Q -> Q y H
# and we still have left-recursion :(

# gamma = x Q, beta = y fails because rewrite
# Q -> y H
# H -> x Q y H
# H  | lambda
# would permit y x y ..., whereas the language permits
# sentences beginning with only x, r or s.

# gamma = lambda, beta = x Q y, new nt H
# notice that with gamma = lambda,
# beta must begin with First(Q)={x}
Q -> x Q y H
H -> x Q y H
   | lambda

# since beta =/=>* lambda,
# must retain the Q -> lambda rule
Q -> lambda

# "R" thought process
# We want _one_ recursive rule, so think of
# R -> R r s t U
# U -> x y | y y y
# now let gamma = r  beta=s t U  new nt V
R -> s t U V
V -> r s t U V
   | lambda
U -> x y | y y y

# We need to permit just R => s t but including
# this base case rule will create a predict set conflict :(
# But we can also permit R => s t by letting
U -> lambda

# The lambda rule for U also permits  s t r s t x y,
# s t r s t x y y
```

Solution: question 3 part b

$$
\begin{array}{rcl}
S & \rightarrow & SUM\ \$ \\
SUM & \rightarrow & PROD\ V \\
V & \rightarrow & plus\ PROD\ V \\
 & | & \lambda \\
PROD & \rightarrow & POWER\ U \\
U & \rightarrow & mult\ POWER\ U \\
 & | & \lambda \\
POWER & \rightarrow & val\ W \\
W & \rightarrow & exp\ POWER \\
 & | & \lambda
\end{array}
$$

```
S -> SUM $

# __ candidate substitutions __ (new terminal V)
# gamma = lambda, beta = plus PROD
# SUM -> plus PROD V
#   V -> plus PROD V
#   V  | lambda
# fails because sentences begin with val ...,
# and this would permit  plus val ...
#
# gamma = plus PROD, beta = lambda
# SUM -> V
# V   -> plus PROD V
#      | lambda
# fails because it would not only permit sentences
# to begin with plus, bit it would also permit empty
# (lambda) sentences into the language
#
# (let's hope this works...)
# gamma = plus, beta = PROD
SUM -> PROD V
  V -> plus PROD V
     | lambda
# since V=>*lambda, we don't need a SUM -> PROD rule anymore.

# The logic for rewriting PROD non-terminals very much
# identical, the symbols simply change from SUM and PROD to
# PROD and POWER (re-written with new non-terminal U)
PROD -> POWER U
   U -> mult POWER U
      | lambda

# For completeness, we need to do common prefix refactoring
# to make this an LL(1) language, using new non-terminal W
POWER -> val W
    W -> exp POWER
       | lambda
```

Solution: question 3 part c

$$
\begin{array}{rcl}
S & \rightarrow & FUNCTIONS\ \$ \\
FUNCTIONS & \rightarrow & FUNCTION\ V \\
V & \rightarrow & FUNCTION\ V \\
 & | & \lambda \\
FUNCTION & \rightarrow & C \\
 & | & P \\
 & | & H \\
C & \rightarrow & type\ id\ oparen\ CPARAMS\ cparen \\
P & \rightarrow & def\ id\ oparen\ PPARAMS\ cparen \\
H & \rightarrow & id\ dblcln\ type\ HPARAMS \\
CPARAMS & \rightarrow & comma\ type\ id\ CPARAMS \\
 & | & \lambda \\
PPARAMS & \rightarrow & comma\ id\ PPARAMS \\
 & | & \lambda \\
HPARAMS & \rightarrow & type\ Q \\
Q & \rightarrow & rarrow\ type\ Q \\
 & | & \lambda
\end{array}
$$

```
S -> FUNCTIONS $

# __ candidate substitutions __ (new terminal V)
# gamma = FUNCTION, beta = lambda
# FUNCTIONS -> V
#         V -> FUNCTION V
#            | lambda
# fails because FUNCTIONS =/=>* lambda in the original grammar, and this will
# permit just that.
#
# gamma = lambda, beta = FUNCTION
# (another hint this is the decomposition we want is that the base
# rule was FUNCTIONS -> FUNCTION, and this often (not always) means
# we want  beta = FUNCTION)
FUNCTIONS -> FUNCTION V
        V -> FUNCTION V
                   | lambda

FUNCTION -> C | P  | H

C -> type id oparen CPARAMS cparen
P -> def id oparen PPARAMS cparen
H -> id dblcln type HPARAMS

# __ candidate substitutions __ (new terminal U)
# gamma = comma type, beta = id
# CPARAMS -> id U
#       U -> comma type id U
#          | lambda
# fails because it permits ... oparen id
# whereas the original grammar requires ... oparen comma type id

# gamma = comma, beta = type id
```

```
# CPARAMS -> type id U
#       U -> comma type id U
#          | lambda
# fails because it permits ... oparen type id
# whereas the original grammar requires ... oparen comma type id

# gamma = lambda, beta = comma type id
# CPARAMS -> comma type id U
#       U -> comma type id U
#       U  | lambda
# Fails as it has predict set conflicts for U

# gamma = comma type id, beta = lambda
# CPARAMS -> U
#       U -> comma type id U
#          | lambda
# This works!  And it hints a simpler re-write not requiring a U:
CPARAMS -> comma type id CPARAMS
           | lambda

# via much the same logic, a simple re-write for PARAMS is
PPARAMS -> comma id PPARAMS
           | lambda

# __ candidate substitutions __ (new terminal Q)
# gamma = lambda, beta = rarrow type
# HPARAMS -> rarrow type Q
#       Q -> rarrow type Q
#          | lambda
# fails as it doesn't permit HPARAMS =>* type
#
# gamma = rarrow type, beta = lambda
# HPARAMS -> Q
#       Q -> rarrow type Q
#          | lambda
# fails as it permits HPARAMS =>* lambda
#
# gamma = rarrow, beta = type
HPARAMS   -> type Q
      Q   -> rarrow type Q
               | lambda
```

4. Write a CFG for a language with two terminals ($\{a, b\}$) that represents all **non-empty** strings that are palindromes. Is your language LL(1)? If not can you refactor it using common prefix or left recursion refactoring so that it is?

Solution:

$$
\begin{array}{lcl}
S & \rightarrow & T\ \$ \\
T & \rightarrow & a\ P\ a \\
 & | & b\ P\ b \\
 & | & a \\
 & | & b \\
P & \rightarrow & a\ P\ a \\
 & | & b\ P\ b \\
 & | & a \\
 & | & b \\
 & | & \lambda
\end{array}
$$

#	$p \in P$	Computed By	Predict Set
1	$S \rightarrow T\ \$$	$FirstSet(RHS)$	a, b
2	$T \rightarrow a\ P\ a$	$FirstSet(RHS)$	a
3	$T \rightarrow b\ P\ b$	$FirstSet(RHS)$	b
4	$T \rightarrow a$	$FirstSet(RHS)$	a
5	$T \rightarrow b$	$FirstSet(RHS)$	b
6	$P \rightarrow a\ P\ a$	$FirstSet(RHS)$	a
7	$P \rightarrow b\ P\ b$	$FirstSet(RHS)$	b
8	$P \rightarrow a$	$FirstSet(RHS)$	a
9	$P \rightarrow b$	$FirstSet(RHS)$	b
10	$P \rightarrow \lambda$	$FollowSet(LHS)$	a, b

	a	b	$
S	1	1	
T	⋆	⋆	
P	⋆	⋆	

$$
\begin{array}{lcl}
S & \rightarrow & T\ \$ \\
T & \rightarrow & a\ TA \\
 & | & b\ TB \\
TA & \rightarrow & P\ a \\
 & | & \lambda \\
TB & \rightarrow & P\ b \\
 & | & \lambda \\
P & \rightarrow & a\ PA \\
 & | & b\ PB \\
 & | & \lambda \\
PA & \rightarrow & P\ a \\
 & | & \lambda \\
PB & \rightarrow & P\ b \\
 & | & \lambda
\end{array}
$$

#	$p \in P$	Computed By	Predict Set
1	$S \rightarrow T\ \$$	$FirstSet(RHS)$	a, b
2	$T \rightarrow a\ TA$	$FirstSet(RHS)$	a
3	$T \rightarrow b\ TB$	$FirstSet(RHS)$	b
4	$TA \rightarrow P\ a$	$FirstSet(RHS)$	a, b
5	$TA \rightarrow \lambda$	$FollowSet(LHS)$	$
6	$TB \rightarrow P\ b$	$FirstSet(RHS)$	a, b
7	$TB \rightarrow \lambda$	$FollowSet(LHS)$	$
8	$P \rightarrow a\ PA$	$FirstSet(RHS)$	a
9	$P \rightarrow b\ PB$	$FirstSet(RHS)$	b
10	$P \rightarrow \lambda$	$FollowSet(LHS)$	a, b
11	$PA \rightarrow P\ a$	$FirstSet(RHS)$	a, b
12	$PA \rightarrow \lambda$	$FollowSet(LHS)$	a, b
13	$PB \rightarrow P\ b$	$FirstSet(RHS)$	a, b
14	$PB \rightarrow \lambda$	$FollowSet(LHS)$	a, b

	a	b	$
S	1	1	
T	2	3	
TA	4	4	5
TB	6	6	7
P	⋆	⋆	
PA	⋆	⋆	
PB	⋆	⋆	