Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: February 14, 2024

QUESTION
1

1 Proving that the grammar is not LL(1)

Given the context-free grammar with the following productions:

$$\begin{aligned}
\mathit{Function} &\to \mathit{Type}\,\mathbf{id}(\mathit{Arguments}) \\
\mathit{Type} &\to \mathbf{id} \\
\mathit{Type} &\to \mathit{Type}\,\mathbf{*} \\
\mathit{Arguments} &\to \mathit{ArgList} \\
\mathit{Arguments} &\to \epsilon \\
\mathit{ArgList} &\to \mathit{Type}\,\mathbf{id,}\mathit{ArgList} \\
\mathit{ArgList} &\to \mathit{Type}\,\mathbf{id}
\end{aligned}$$

It can be easily observed that the grammar has left recursion ($Type \to Type\mathbf{*}$) and cannot be parsed by a top-down parser and is thus not LL(1).

We can also show that the above grammar is not LL(1) by constructing the predictive parsing table M and observing that there is some entry in the table which corresponds to more than one possible productions. We first start by computing the FIRST and FOLLOW sets.

Computing FIRST and FOLLOW Sets

FIRST Sets

- FIRST($\mathbf{T}$) = $\{\mathbf{T}\}$ where T are all the terminal symbols i.e. id()*,$ and ϵ.
- FIRST(Type) = $\{\mathbf{id}\}$ as we have the production $\mathit{Type} \to \mathbf{id}$
- FIRST($\mathit{ArgList}$) = FIRST(Type) = $\{\mathbf{id}\}$
- FIRST($\mathit{Arguments}$) = FIRST($\mathit{ArgList}$) $\cup \{\epsilon\} = \{\mathbf{id}, \epsilon\}$
- FIRST($\mathit{Function}$) = FIRST(Type) = $\{\mathbf{id}\}$

FOLLOW Sets

- FOLLOW($\mathit{Function}$) = $\{\$\}$
- FOLLOW(Type) = $\{\mathbf{id}, \mathbf{*}\}$ since Type is followed by id and *.
- FOLLOW($\mathit{Arguments}$) = $\{\mathbf{)}\}$
- FOLLOW($\mathit{ArgList}$) = $\{\mathbf{)}\}$ since we have the production $\mathit{Arguments} \to \mathit{ArgList}$.

Constructing the Predictive Parsing Table M

We fill the table entries M[A, a] for a nonterminal A and terminal a with the production $A \to \alpha$ if:

- $a \in \text{FIRST}(\alpha)$, or
- $\epsilon \in \text{FIRST}(\alpha)$ and $a \in \text{FOLLOW}(A)$.

NON-TERMINAL	**INPUT SYMBOL**					
	id	*	(	)	,	$
Function	*Function* $\to$ *Type***id** (*Arguments*)					
Type	*Type* $\to$ **id** *Type* $\to$ *Type**					
Arguments	*Arguments* $\to$ *ArgList*			*Arguments* $\to \epsilon$		
ArgList	*ArgList* $\to$ *Type***id**, *ArgList* *Arglist* $\to$ *Type***id**					

Table 1: Predictive Parsing Table for the Given CFG

As it can be evidently seen, the table doesn't have a unique production associated for M[*Type*,**id**] and M[*ArgList*, **id**]. Thus it is not LL(1).

2 Constructing the LL(1) grammar

We first eliminate left-recursion in the grammar. The left-recursive derivation is *Type*$\to$*Type**. We introduce a new non-terminal $Type_1$ and the productions are as follows:

$$\begin{aligned}
Function &\to Type\,\mathbf{id}(Arguments) \\
Type &\to \mathbf{id}\,Type_1 \\
Type_1 &\to *\,Type_1 \mid \epsilon \\
Arguments &\to ArgList \mid \epsilon \\
ArgList &\to Type\,\mathbf{id} \mid Type\,\mathbf{id,}ArgList
\end{aligned}$$

Now, we perform left-factoring to remove the ambiguity about which production to use from *ArgList*. For this purpose, we introduce another non-terminal $ArgList_1$ and the productions are as follows:

$$
\begin{aligned}
\textit{Function} &\rightarrow \textit{Type}\,\mathbf{id}\,\mathbf{(}\textit{Arguments}\mathbf{)} \\
\textit{Type} &\rightarrow \mathbf{id}\,\textit{Type}_1 \\
\textit{Type}_1 &\rightarrow \mathbf{*}\,\textit{Type}_1 \mid \epsilon \\
\textit{Arguments} &\rightarrow \textit{ArgList} \mid \epsilon \\
\textit{ArgList} &\rightarrow \textit{Type}\,\mathbf{id}\,\textit{ArgList}_1 \\
\textit{ArgList}_1 &\rightarrow \epsilon \mid \mathbf{,}\textit{ArgList}
\end{aligned}
$$

The resulting grammar should be LL(1).

3 FIRST and FOLLOW sets for the transformed grammar

FIRST Sets

- FIRST($\mathbf{T}$) = $\{\mathbf{T}\}$ for all terminals i.e. id()*,$ and ϵ.
- FIRST($\textit{Type}$) = $\{\mathbf{id}\}$ = FIRST($\textit{Function}$) = FIRST($\textit{ArgList}$)
- FIRST($\textit{Type}_1$) = $\{\mathbf{*}, \epsilon\}$
- FIRST($\textit{Arguments}$) = $\{\mathbf{id}, \epsilon\}$
- FIRST($\textit{ArgList}_1$) = $\{\mathbf{,}, \epsilon\}$

FOLLOW Sets

- FOLLOW($\textit{Function}$) = $\{\$\}$
- FOLLOW($\textit{Type}$) = $\{\mathbf{id}\}$
- FOLLOW($\textit{Type}_1$) = $\{\mathbf{id}\}$
- FOLLOW($\textit{Arguments}$) = $\{\mathbf{)}\}$
- FOLLOW($\textit{ArgList}$) = $\{\mathbf{)}\}$
- FOLLOW($\textit{ArgList}_1$) = $\{\mathbf{)}\}$

4 Predictive Parsing Table for transformed grammar

NON-TERMINAL	**INPUT SYMBOL**					
	id	*	(	)	,	$
Function	$Function \rightarrow Type\mathbf{id}(Arguments)$					
Type	$Type \rightarrow \mathbf{id}\, Type_1$					
$Type_1$	$Type_1 \rightarrow \epsilon$	$Type_1 \rightarrow *Type_1$				
Arguments	$Arguments \rightarrow ArgList$			$Arguments \rightarrow \epsilon$		
ArgList	$ArgList \rightarrow Type\mathbf{id}ArgList_1$					
$ArgList_1$				$ArgList_1 \rightarrow \epsilon$	$ArgList_1 \rightarrow ,ArgList$	

Table 2: Predictive Parsing Table for the Transformed CFG

As it can be seen, each cell in the table corresponds to a unique transition thus we have a LL(1) grammar.

Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: February 14, 2024

QUESTION
2

We have been given the CFG

$$
\begin{aligned}
\text{S} &\rightarrow \text{LM} \mid \text{Lp} \mid \text{qLr} \mid \text{sr} \mid \text{qsp} \\
\text{L} &\rightarrow \text{aMb} \mid \text{s} \mid \text{t} \\
\text{M} &\rightarrow \text{t}
\end{aligned}
$$

We first augment this grammar by adding the non-terminal S' s.t.

$$
\begin{aligned}
\text{S'} &\rightarrow \text{S} \\
\text{S} &\rightarrow \text{LM} \mid \text{Lp} \mid \text{qLr} \mid \text{sr} \mid \text{qsp} \\
\text{L} &\rightarrow \text{aMb} \mid \text{s} \mid \text{t} \\
\text{M} &\rightarrow \text{t}
\end{aligned}
$$

1 Is the CFG SLR(1)?

FIRST and FOLLOW Sets

FIRST Sets

- FIRST(**T**) = {**T**} for all terminals T $\in$ {a, b, p, q, r, s, t, \$}
- FIRST(M) = {t}
- FIRST(L) = {a, s, t}
- FIRST(S) = FIRST(L) $\cup$ {q, s} = {a, q, s, t} (which is also FIRST(S'), note that S' was simply introduced for augmentation)

FOLLOW Sets

- FOLLOW(S) = {\$} [which is also FOLLOW(S')]
- FOLLOW(L) = FIRST(M) $\cup$ {p, r} = {p, r, t} (since we have S $\rightarrow$LM | Lp | qLr)
- FOLLOW(M) = FOLLOW(S) $\cup$ {b} = {\$, b} (since we have S $\rightarrow$ LM and L $\rightarrow$ aMb)

LR(0) canonical collection

We begin with the computation of CLOSURE({[S' → .S]}) using the definition of CLOSURE. It results in the item set:

S' → .S
S → .LM
S → .Lp
S → .qLr
S → .sr
S → .qsp
L → .aMb
L → .s
L → .t

Similarly, we iteratively build the item sets using the GOTO function and iterating over all grammar symbols to obtain the LR(0) canonical collection:

I_0
$S' \to .S$
$S \to .LM$
$S \to .Lp$
$S \to .qLr$
$S \to .sr$
$S \to .qsp$
$L \to .aMb$
$L \to .s$
$L \to .t$

I_1
$S' \to S.$

I_7
$S \to LM.$

I_2
$S \to L.M$
$S \to L.p$
$M \to .t$

I_8
$S \to Lp.$

I_3
$S \to q.Lr$
$S \to q.sp$
$L \to .aMb$
$L \to .s$
$L \to .t$

I_{10}
$S \to qL.r$

I_{14}
$S \to qLr.$

I_9
$M \to t.$

I_{11}
$S \to qs.p$
$L \to s.$

I_{15}
$S \to qsp.$

I_6
$L \to t.$

I_4
$S \to s.r$
$L \to s.$

I_5
$L \to a.Mb$
$M \to .t$

I_{12}
$S \to sr.$

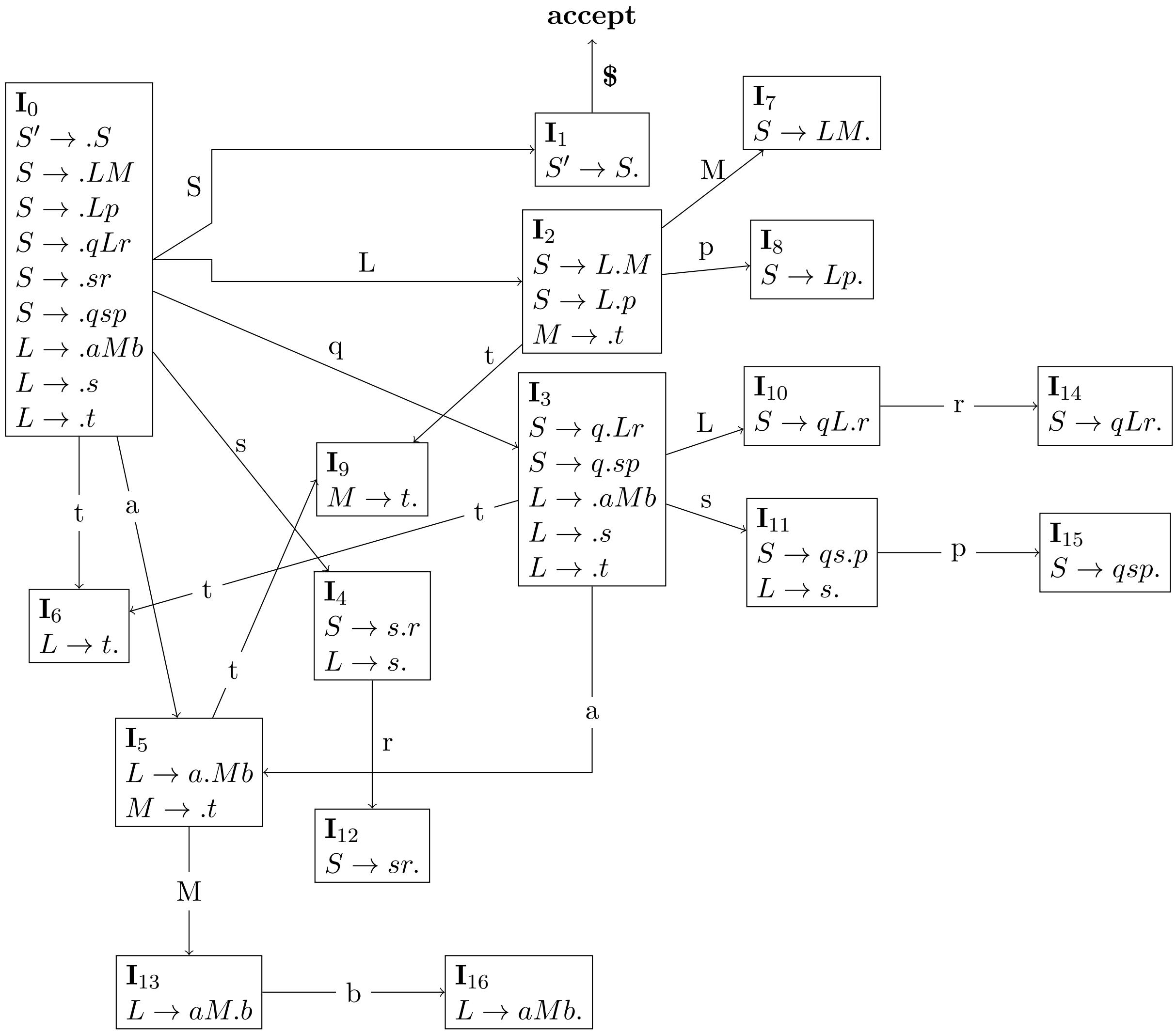
I_{13}
$L \to aM.b$

I_{16}
$L \to aMb.$

This is the LR(0) canonical collection. To construct the LR(0) automaton:

The states of the automaton are coded as the sets of items from the LR(0) canonical collection and the transitions are given by the GOTO function, with the start state being CLOSURE({[S' → .S]}). We introduce the transitions by considering the definition of the GOTO function:

GOTO(I,X) is defined as the closure of the set of all items [A → $\alpha X.\beta$] such that [A → $\alpha .X\beta$] is in I, where I is a set of items and X is a grammar symbol. Thus we have this automaton:



Here, the start state corresponds to $\mathbf{I}_0$.

SLR Parsing Table

Using the LR(0) canonical items and the GOTO function and FOLLOW(A) for all non-terminals A, we determine the parsing actions for each state (where state i corresponds to I_i).

In the table, si corresponds to "shift j" and ri corresponds to using the ith reduction (the numbers associated with each production are defined below the table). **acc** means the accepting state and all empty entries correspond to error cases.

STATE	ACTION								GOTO		
	a	b	p	q	r	s	t	$	S	L	M
0	s5			s3		s4	s6		1	2	
1								**acc**			
2			s8				s9				7
3	s5					s11	s6			10	
4			r8		s12 or r8		r8				
5							s9				13
6			r9		r9		r9				
7								r2			
8								r3			
9		r10						r10			
10					s14						
11			s15 or r8		r8		r8				
12								r5			
13		s16									
14								r4			
15								r6			
16			r7		r7		r7				

Table 3: SLR Parsing Table for the CFG

$$
\begin{aligned}
S' &\rightarrow S \ (1) \\
S &\rightarrow LM\ (2) \mid Lp\ (3) \mid qLr\ (4) \mid sr\ (5) \mid qsp\ (6) \\
L &\rightarrow aMb\ (7) \mid s\ (8) \mid t\ (9) \\
M &\rightarrow t\ (10)
\end{aligned}
$$

Here, the numbers in the bracket are used to number each production to denote it compactly in the parsing table.

As it can be clearly seen from the table, the grammar is not SLR(1) since it cannot take a unique parsing decision on the input r when in state 4 or on the input p when in state 11. This is because SLR doesn't have enough mechanisms to remember about what it has previously read to reach the current state.

2 Is the CFG LALR(1)?

The augmented grammar remains the same.

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow LM \mid Lp \mid qLr \mid sr \mid qsp \\
L &\rightarrow aMb \mid s \mid t \\
M &\rightarrow t
\end{aligned}
$$

FIRST and FOLLOW Sets

FIRST Sets

- FIRST(**T**) = {**T**} for all terminals T ∈ {a, b, p, q, r, s, t, \$}
- FIRST(M) = {t}
- FIRST(L) = {a, s, t}
- FIRST(S) = FIRST(L) ∪ {q, s} = {a, q, s, t} (which is also FIRST(S'), note that S' was simply introduced for augmentation)

FOLLOW Sets

- FOLLOW(S) = {\$} [which is also FOLLOW(S')]
- FOLLOW(L) = FIRST(M) ∪ {p, r} = {p, r, t} (since we have S →LM | Lp | qLr)
- FOLLOW(M) = FOLLOW(S) ∪ {b} = {\$, b} (since we have S → LM and L → aMb)

LALR collection

We first construct the LR(1) collection of items based on the definitions of CLOSURE and GOTO functions:

I_0
$S' \to .S$, \$
$S \to .LM$, \$
$S \to .Lp$, \$
$S \to .qLr$, \$
$S \to .sr$, \$
$S \to .qsp$, \$
$L \to .aMb$, p/t
$L \to .s$, p/t
$L \to .t$, p/t

I_1
$S' \to S.$, \$

I_2
$S \to L.M$, \$
$S \to L.p$, \$
$M \to .t$, \$

I_3
$L \to a.Mb$, p/t
$M \to .t$, b

I_4
$S \to q.Lr$, \$
$S \to q.sp$, \$
$L \to .aMb$, r
$L \to .s$, r
$L \to .t$, r

I_5
$S \to s.r$, \$
$L \to s.$, p/t

I_6
$L \to t.$, p/t

I_7
$S \to LM.$, \$

I_8
$S \to Lp.$, \$

I_9
$M \to t.$, \$

I_{10}
$L \to aM.b$, p/t

I_{11}
$M \to t.$, b

I_{12}
$S \to qL.r$, \$

I_{13}
$L \to a.Mb$, r
$M \to .t$, b

I_{14}
$S \to qs.p$, \$
$L \to s.$, r

I_{15}
$L \to t.$, r

I_{16}
$S \to sr.$, \$

I_{17}
$L \to aMb.$, p/t

I_{18}
$S \to qLr.$, \$

I_{19}
$L \to aM.b$, r

I_{20}
$S \to qsp.$, \$

I_{21}
$L \to aMb.$, r

Given the algorithmic constructon of the LALR collection of states by merging the common cores, we have the following collection:

$\mathbf{I}_1$
$S' \rightarrow S.$, \$

$\mathbf{I}_7$
$S \rightarrow LM.$, \$

$\mathbf{I}_0$
$S' \rightarrow .S$, \$
$S \rightarrow .LM$, \$
$S \rightarrow .Lp$, \$
$S \rightarrow .qLr$, \$
$S \rightarrow .sr$, \$
$S \rightarrow .qsp$, \$
$L \rightarrow .aMb$, p/t
$L \rightarrow .s$, p/t
$L \rightarrow .t$, p/t

$\mathbf{I}_2$
$S \rightarrow L.M$, \$
$S \rightarrow L.p$, \$
$M \rightarrow .t$, \$

$\mathbf{I}_8$
$S \rightarrow Lp.$, \$

$\mathbf{I}_{17,21}$
$L \rightarrow aMb.$, p/r/t

$\mathbf{I}_{9,11}$
$M \rightarrow t.$, b/\$

$\mathbf{I}_{3,13}$
$L \rightarrow a.Mb$, p/r/t
$M \rightarrow .t$, b

$\mathbf{I}_{10,19}$
$L \rightarrow aM.b$, p/r/t

$\mathbf{I}_{18}$
$S \rightarrow qLr.$, \$

$\mathbf{I}_{6,15}$
$L \rightarrow t.$, p/r/t

$\mathbf{I}_4$
$S \rightarrow q.Lr$, \$
$S \rightarrow q.sp$, \$
$L \rightarrow .aMb$, r
$L \rightarrow .s$, r
$L \rightarrow .t$, r

$\mathbf{I}_{12}$
$S \rightarrow qL.r$, \$

$\mathbf{I}_5$
$S \rightarrow s.r$, \$
$L \rightarrow s.$, p/t

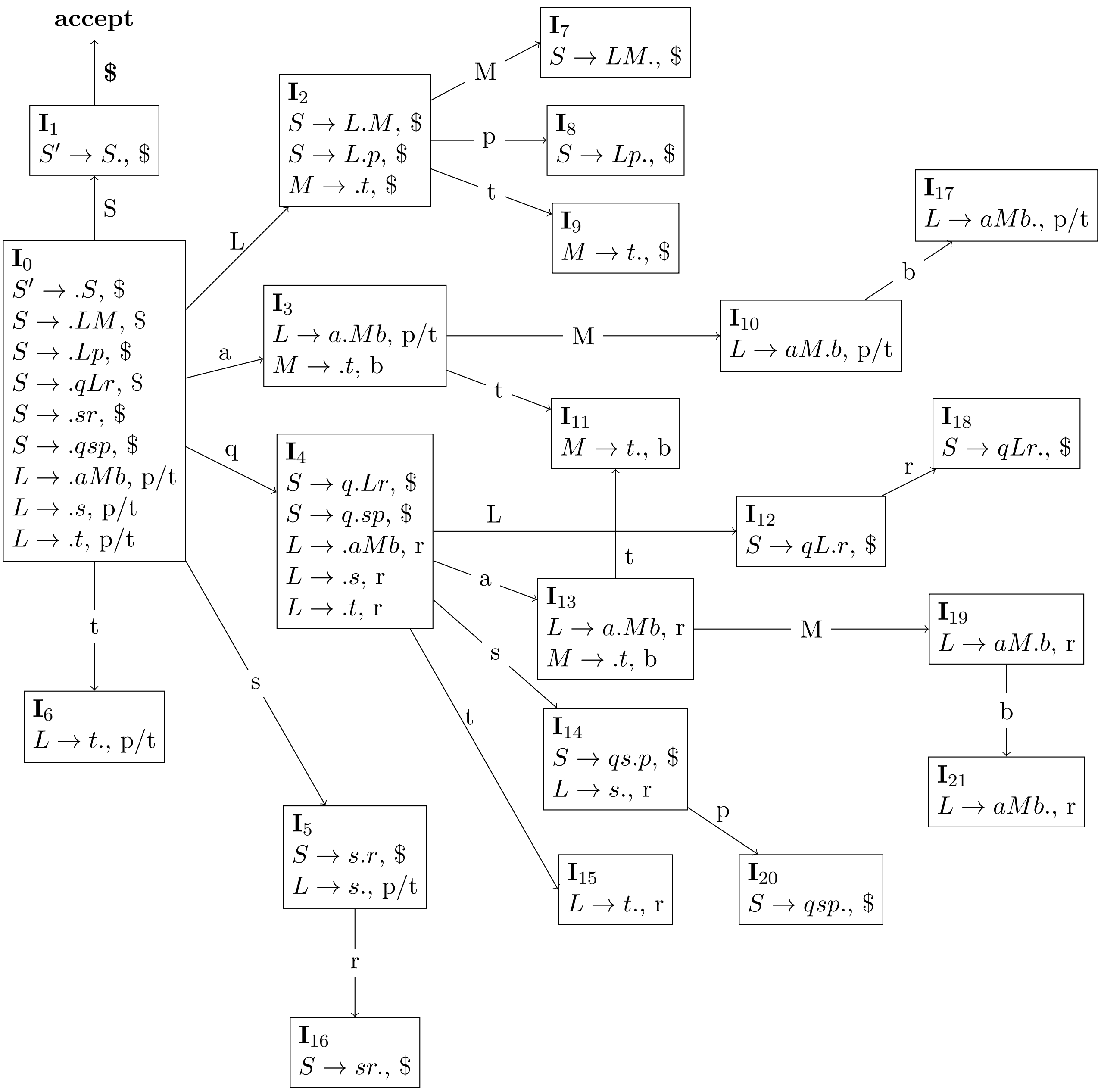
$\mathbf{I}_{14}$
$S \rightarrow qs.p$, \$
$L \rightarrow s.$, r

$\mathbf{I}_{20}$
$S \rightarrow qsp.$, \$

$\mathbf{I}_{16}$
$S \rightarrow sr.$, \$

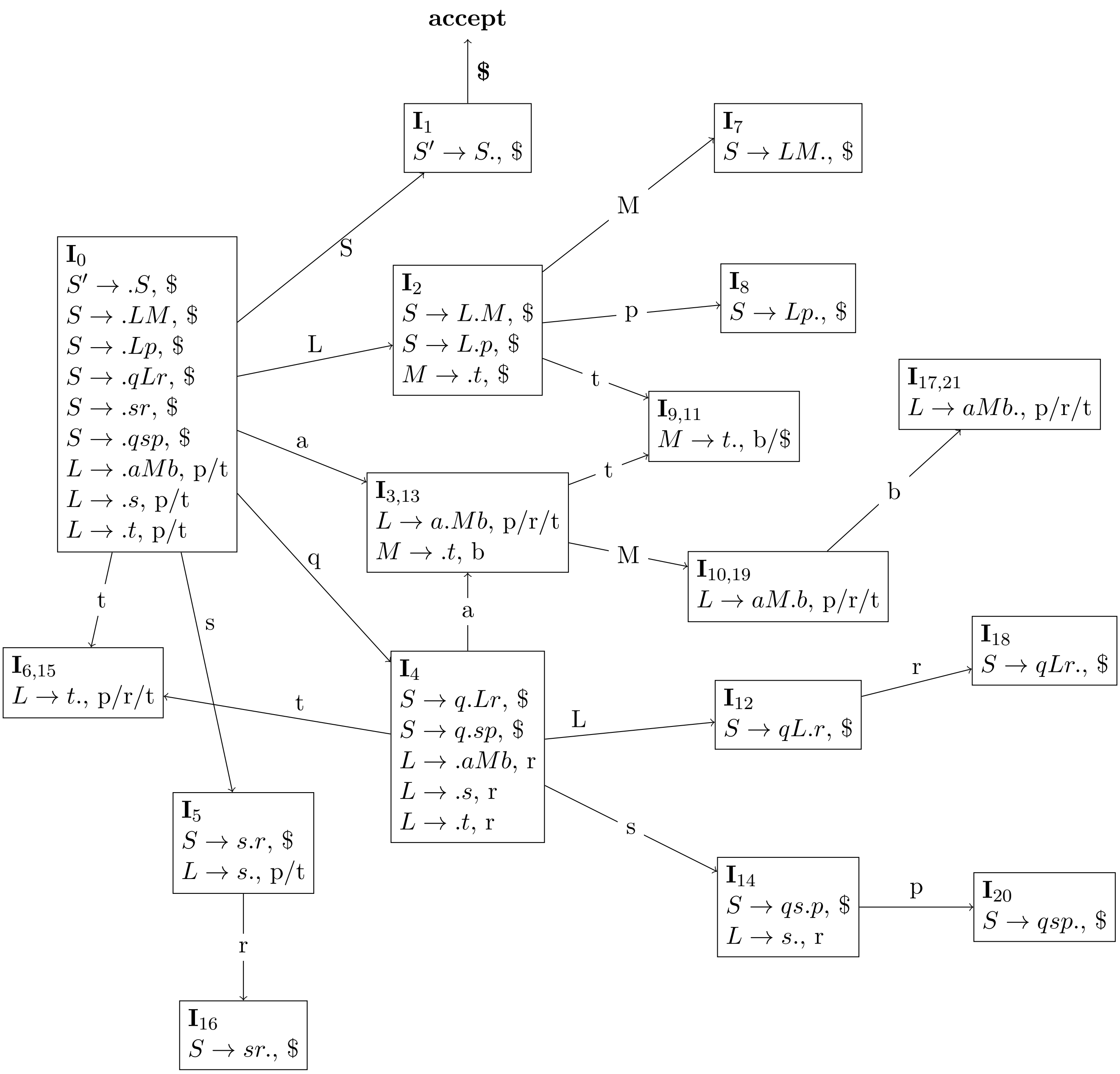
LR(1) automaton

We add the GOTO transitions to the above collection of atoms to get the automaton. The start state corresponds to CLOSURE({[S' → .S]}), which has been marked as $\mathbf{I}_0$ in the diagram.



LALR(1) Parsing Table

Now that we have the collection of items, we need the ACTION and GOTO values for the states. For the sake of brevity, the automaton with the reduced sets is also drawn:



The transitions have been numbered as follows:

S' → S (1)
S → LM (2) | Lp (3) | qLr (4) | sr (5) | qsp (6)
L → aMb (7) | s (8) | t (9)
M → t (10)

The ACTION and GOTO values are computed as per the algorithm and displayed below. 'si' refers to "shift i", 'ri' refers to using the ith transition as a reduction. **acc** refers to the accept action. All the blank entries become error cases.

STATE	ACTION								GOTO		
	a	b	p	q	r	s	t	$	S	L	M
0	s3,13			s4		s5	s6,15		1	2	
1								**acc**			
2			s8				s9,11				7
3,13							s9,11				10,19
4	s3,13					s14	s6,15			12	
5			r8		s16		r8				
6,15			r9		r9		r9				
7								r2			
8								r3			
9,11		r10						r10			
10,19		s17,21									
12					s18						
14			s20		r8						
16								r5			
17,21			r7		r7		r7				
18								r4			
20								r6			

Table 4: LALR Parsing Table for the CFG

As it can be seen, for each state on each input, we can take a unique parsing decision. Thus the grammar is LALR(1).

Compiler Design (CS335), Spring 2024
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: February 14, 2024

QUESTION

3

1 Problem Description

The goal was the generate a parser and display the desired output for the given markup schema.

2 Instructions to Run

The subdirectory 'problem3' contains 3 files: prob3.l, prob3.y and prob3.sh. To run, ensure that the present working directory is problem3. Now,

- Open prob3.sh. It contains a variable called $file\_name$. Provide the relative file path for the testcase as the value for this variable. Ensure that the entire filename (with the file extension) is written. Eg. if the filename is *test.html*, this entire name should be written.
- Now, run prob3.sh. It can be executed in 2 ways:
  - ./prob3.sh: It displays the output on the terminal (stdout)
  - ./prob3.sh -f: It redirects the output to file_name.output. Note that the .output file will be in the same subdirectory as the original testcase.
- The script automatically deletes all the additional files that it has created.

3 Corner/Error Cases and Format of Output

- The parser continues parsing the input till it encounters the first error. It reports the error and terminates the execution. All misspelt/random tags as well as words outside the placeholder for a given text are ignored.
- The output format is as instructed in the PDF.
- The parser detects the following errors:
  - Marks are out of range for the given question type.
  - Marks are not enclosed within ".
  - There are too few/many choices.
  - If there are 0 correct choices for a question.
  - If there are >1 correct choices for a singleselect type question.
  - Number of correct choices > number of choices for a multiselect type question.
  - Missing opening and closing tags for quiz/singleselect/multiselect/choice/correct. Note that stray closing tags are also highlighted by reporting that the opening tag is missing!

- The following forms of input required modifications to the grammar and in my opinion are worthy of being included in the testcases:
  - **An empty program**: An empty file should also be syntactically valid and thus accepted by the parser without reporting any error.
  - **An integer present outside the placeholder**: The following input required non-trivial modifications to the grammar to parse correctly. Note that it is syntactically correct and should result in no errors:

    ```
    <quiz>
        <singleselect marks="2"> This is a question
            <choice>20</choice>10
            <choice>15</choice>
            <choice>10</choice>
            <correcct>15</correct>
        </singleselect>
    </quiz>
    ```

    i.e. the additional 10 (a **number**) present outside the tokens shouldn't result in an error. Along similar lines, the input:

    ```
    <quiz>
        <singleselect marks=" 1"> This is a question
            <choice>20</choice>10
            <choice>15</choice>
            <choice>10</choice>12
            15</correct>
        </singleselect>
    </quiz>
    ```

    should result in a 'missing opening correct tag' error. This also required some modification to the grammar for correct error reporting.
  - **Attribute inside any of the valid tag**: As instructed in the PDF, we have been told that any attribute except marks inside any placeholder has to be ignored. Thus an input of the following type:

    ```
    <quiz ignore this text>
        <singleselect marks=" 2"> This is a question
            <choice>A</choice>
            <choice asdasd>B</choice>
            <choice>C</choICE>
        </singleselect>
    </quiz>
    ```

    Should not result in any error.