Consider the grammar
G = ({S, T}, {a, b, c}, S, P),
where P is:

1. **S : SaSb**
2. **S : Sa**
3. **S : Tc**
4. **T : cT**
5. **T :** ε **//the empty rule**

Convert this to a right recursive grammar.

Compute the **FIRST** and **FOLLOW** sets for your new grammar's non-terminals (you may have grown one when converting left to right recursion).

**FIRST( ) = { } FOLLOW( ) = { }**

**FIRST( ) = { } FOLLOW( ) = { }**

**FIRST( ) = { } FOLLOW( ) = { }**

Produce the **LL(1)** parsing table based on these sets. Fill in production numbers, not productions. If there are any conflicts, circle those entries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **$** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Consider the following context free grammar

1. **S' : S**
2. **S : A'a'**
3. **S : B'b'B'a'**
4. **A : 'a'**
5. **B : 'a'**

 (a) Produce the description of an **SLR(1)** parser. There are no more than 10 states. I already did 3 of them. Point out any and all conflicts that arise.

# State Item(s) Action Goto

**0 S' : \_S S: goto 1**

 **S : \_Aa A: goto 2**

 **S : \_BbBa B: goto 3**

 **A : \_a a: shift 4**

 **B : \_a**

**1 S' : S\_ $: reduce 0 (accept)**

**2 S: A\_a a shift 5**

Once again, consider the following context free grammar

1. **S' : S**
2. **S : A'a'**
3. **S : B'b'B'a'**
4. **A : 'a'**
5. **B : 'a'**

 (b) Produce the description of an **LR(1)** parser. There are no more than 10 states. I already did 3 of them. Point out any and all conflicts that might arise.

# State Item(s) Action Goto

**0 S' : \_S , $ S: goto 1**

 **S : \_Aa , $ A: goto 2**

 **S : \_BbBa , $ B: goto 3**

 **A : \_a , a a: shift 4**

 **B : \_a , b**

**1 S' : S\_ , $ $: reduce 0 (accept)**

**2 S: A\_a , $ a shift 5**

(c) Can a conflict arise if we produce an LALR(1) parser for this grammar. If so, where? If not, why not?

 Consider the grammar
G = ({S, T}, {a, b, c}, S, P),
where P is:

1. **S : SaSb**
2. **S : Sa**
3. **S : Tc**
4. **T : cT**
5. **T :** ε **//the empty rule**

Convert this to a right recursive grammar.

1. ***S : T c S’***
2. ***S’: a S b S’***
3. ***S’: aS’***
4. ***S’:*** ε
5. ***T: cT***
6. ***T:*** ε

Compute the **FIRST** and **FOLLOW** sets for your new grammar's non-terminals (you may have grown one when converting left to right recursion).

**FIRST( S ) = { *c* } FOLLOW( S ) = { *$, b* }**

**FIRST( S’ ) = { *a,*** *ε* **} FOLLOW( S’ ) = { *$, b* }**

**FIRST( T ) = { *c,*** *ε* **} FOLLOW( T ) = { *c* }**

Produce the **LL(1)** parsing table based on these sets. Fill in production numbers, not productions. If there are any conflicts, circle those entries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **$** |
| ***S*** |  |  | ***1*** |  |
| ***S’*** | ***2,3*** | ***4*** |  | ***4*** |
| ***T*** |  |  | ***5,6*** |  |

Consider the following context free grammar

1. **S' : S *FOLLOW[S’] = {$}***
2. **S : A a *FOLLOW[S] = {$}***
3. **S : B b B a**
4. **A : a *FOLLOW[A] = {a}***
5. **B : a *FOLLOW[B] = {a,b}***

 (a) Produce the description of an **SLR(1)** parser. There are no more than 10 states. I already did 3 of them. Point out any and all conflicts that arise.

# State Item(s) Action Goto

**0 S' : \_S S: goto 1**

 **S : \_Aa A: goto 2**

 **S : \_BbBa B: goto 3**

 **A : \_a a: shift 4**

 **B : \_a**

**1 S' : S\_ $: reduce 0 (accept)**

**2 S: A\_a a shift 5**

***3 S: B\_bBa b shift 6***

***4 A: a\_ a reduce 3***

 ***B: b\_ a,b reduce 4***

***5 S: Aa\_ $ reduce 1***

***6 S: Bb\_Ba B: goto 7***

 ***B: \_a a shift 8***

***7 S: BbB\_a a shift 9***

*8* ***B: a\_ a,b reduce 4***

***9 S: BbBa\_ $ reduce 2***

Once again, consider the following context free grammar

1. **S' : S**
2. **S : A'a'**
3. **S : B'b'B'a'**
4. **A : 'a'**
5. **B : 'a'**

 (b) Produce the description of an **LR(1)** parser. There are no more than 10 states. I already did 3 of them. Point out any and all conflicts that might arise.

# State Item(s) Action Goto

**0 S' : \_S , $ S: goto 1**

 **S : \_Aa , $ A: goto 2**

 **S : \_BbBa , $ B: goto 3**

 **A : \_a , a a: shift 4**

 **B : \_a , b**

**1 S' : S\_ , $ $: reduce 0 (accept)**

**2 S: A\_a , $ a shift 5**

***3 S: B\_bBa ,$ b shift 6***

***4 A: a\_ ,a a reduce 3***

 ***B: b\_ ,b b reduce 4***

***5 S: Aa\_ ,$ $ reduce 1***

***6 S: Bb\_Ba ,$ B: goto 7***

 ***B: \_a ,a a shift 8***

***7 S: BbB\_a ,$ a shift 9***

*8* ***B: a\_ ,a a reduce 4***

***9 S: BbBa\_ ,$ $ reduce 2***

 c) Can a conflict arise if we produce an LALR(1) parser for this grammar. If so, where? If not, why not?

***No conflict can arise because the cores of each state are distinct, so we will merge no lookaheads when we convert to LALR(1). In fact, the LR(1) and LALR(1) parsers for this grammar are identical.***