Lab 3

Answer 1

Given grammer:

```
A → aA | aB
B → Ab | Bb | C
C → c | Dc | Ac`
D → Bd | d | Dd
```

There exists simple left recursion (immediate) at:

1. `B -> Bb` - B directly calls itself at the first symbol in the production `B → Ab | Bb | C`
2. `D -> Dd` - D also directly calls itself at `D → Bd | d | Dd`

There also exists general left recursion (indirect) at:

1. `B → Ab | Bb | C` - `B` directly calls `C` which (via. grammer `C → c | Dc | Ac`) calls `D` at `C -> Dc` and finally calls `B` back again (via. `D → Bd | d | Dd`). Resulting in an indirect recursion via calls: `B -> C -> D -> B`.
2. `C → c | Dc | Ac` - `C` directly calls `D` which calls `B`(via. grammer `D → Bd | d | Dd`). `B` then directly calls `C` again (via. grammer `D → Bd | d | Dd`).
3. `D → Bd | d | Dd` - `D` directly calls `B` which calls `C` (via. grammer `B → Ab | Bb | C`) which then again calls `C` that calls `D` again. (via. grammer `C → c | Dc | Ac`).

Answer 2

Given grammer:

```
A → aA | aB
B → Ab | Bb | C
C → c | Dc | Ac
D → Bd | d | Dd
```

Step 1:

Order the list of all non-terminals in as close to top-down order as possible. The current order already seems to align with this.

```
A → aA | aB
B → Ab | Bb | C
```

```
C → c | Dc | Ac
D → Bd | d | Dd
```

Step 2:

Process each non-terminal

1. `A → aA | aB` remains unchanged since it already starts with a terminal and no earlier non-terminals exist. No substitution needed.
2. `B → Ab | Bb | C`

`B → Ab` starts with A (non-terminal) on the RHS which needs to be substituted

`B –> aAb | aBb | Bb | C` has left recursion and follows the pattern S → Sα1 | ... | Sαn | β1 | ... | βn. Upon tranformation yeilds `B → aAb B' | aBb B' | C B'` & `B' → b B' | ε`

```
A → aA | aB
B → aAb B' | aBb B' | C B'
B' → b B' | ε
C → c | Dc | Ac
D → Bd | d | Dd
```

3. `C → c | Dc | Ac` starts with a non-terminal at `D` and `A` which needs to be substituted with the right values for `A`

Yeilds `C → c | Dc | aAc | aBc` which only have one non-terminal variable start i.e. at `D` but that is in the next step.

```
A → aA | aB
B → aAb B' | aBb B' | C B'
B' → b B' | ε
C → c | Dc | aAc | aBc
D → Bd | d | Dd
```

4. `D → Bd | d | Dd` starts with a non-terminal at `Bd` and `Dd` (itself) which needs to be removed.

`D → aAb B'd | aBb B'd | C B'd | d | Dd` - Substibuting the value of `B` (non-terminal start) `D → aAb B'd | aBb B'd | cB'd | DcB'd | AcB'd | d | Dd` - Substibuting the value of `C` (non-terminal start) `D → aAb B'd | aBb B'd | cB'd | DcB'd | aAcB'd | aBcB'd | d | Dd` - Substibuting the value of `A` (non-terminal start)

This new production grammer has no items that start with a non-terminal (other than `D`) and which yeild the result that looks of the pattern S → Sα1 | ... | Sαn | β1 | ... | βn.

- Productions starting with D:

```
D → DcB'd
D → Dd
```

- Productions not starting with D:

```
D → aAb B'd
D → aBb B'd
D → cB'd
D → aAcB'd
D → aBcB'd
D → d
```

- All β productions (not starting with D):

```
β₁ = aAb B'd
β₂ = aBb B'd
β₃ = cB'd
β₄ = aAcB'd
β₅ = aBcB'd
β₆ = d
```

- All α productions (starting with D)

```
α₁ = cB'd
α₂ = d
```

- Apply transformation:

```
D → β₁ D' | β₂ D' | β₃ D' | β₄ D' | β₅ D' | β₆ D'
D' → α₁ D' | α₂ D' | ε
```

- Substituting the values

```
D → aAb B'd D' | aBb B'd D' | cB'd D' | aAcB'd D' | aBcB'd D' | d D'
D' → cB'd D' | d D' | ε
```

Final Grammer:

```
A → aA | aB
B → aAb B' | aBb B' | C B'
```

```
B' → b B' | ε
C → c | Dc | aAc | aBc
D → aAb B'd D' | aBb B'd D' | cB'd D' | aAcB'd D' | aBcB'd D' | d D'
D' → cB'd D' | d D' | ε
```