Worksheet # 21 Solution

(From Lecture #21 given on 4/8/2019)

1. Consider the following left recursive grammar.

$$S \to Aa \mid b$$
$$A \to Ac \mid Sd \mid \epsilon$$

Convert the above grammar to one that is suitable for LL parsing by removing left recursion from it.

- **The grammar can be represented as a 4-tuple. G = (S, NT, T, P)**

S is the start symbol
NT = **{S, A}** (Set of non-terminal variables)
T = **{a, b, c, d}** (Set of terminal variables)
P = **{S ⇒ Aa, S ⇒ b, A ⇒ Ac, A ⇒ Sd, A ⇒ ε}** (Set of productions)

1. Consider the following left recursive grammar.

$$S \rightarrow Aa \mid b$$
$$A \rightarrow Ac \mid Sd \mid \epsilon$$

Convert the above grammar to one that is suitable for LL parsing by removing left recursion from it.

- **A grammar is left recursive if ∃ A ∈ NT such that ∃ a derivation A ⇒+ Aα, for some string α ∈ (NT ∪ T)+**

- **S** ⇒ Aa ⇒ **S**da (indirect left recursion)
- **A** ⇒ **A**c (direct left recursion)

Algorithm to remove left recursion

- The algorithm removes left recursion via 2 techniques.
  1. **convert <u>indirect left recursion</u> to <u>direct left recursion</u>**
  2. **Rewrite <u>direct left recursion</u> as <u>right recursion</u>**

```
impose an order on the nonterminals, A1, A2, ..., An

for i ← 1 to n do;
    for j ← 1 to i - 1 do;
        if ∃ a production Ai → Aj γ
            then replace Ai → Aj γ with one or more
                productions that expand Aj
    end;
    rewrite the productions to eliminate
        any direct left recursion on Ai
end;
```

1

2

Using the algorithm : 1st iteration of outer loop

- Let's impose the order of non-terminals : **S, A**

```
impose an order on the nonterminals, A1, A2, ..., An

for i ← 1 to n do;
    for j ← 1 to i - 1 do;
        if ∃ a production Ai→Ajγ
            then replace Ai→Ajγ with one or more
                 productions that expand Aj
    end;
    rewrite the productions to eliminate
         any direct left recursion on Ai
end;
```

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \epsilon$$

Nothing is done in the inner loop.
No direct recursion on S exists.
No change is made to the grammar.

Using the algorithm : 2nd iteration of outer loop

- Imposed order of non-terminals : **S, A**

impose an order on the nonterminals, $A_1, A_2, \ldots, A_n$

```
for i ← 1 to n do;
    for j ← 1 to i - 1 do;
        if ∃ a production Ai → Aj γ
            then replace Ai → Aj γ with one or more
                productions that expand Aj
    end;
    rewrite the productions to eliminate
        any direct left recursion on Ai
end;
```

S ⇒ Aa | b

A ⇒Ac | **S**d | ε

In the inner loop, production **A ⇒ Sd** is

replaced with

A ⇒Aad, A ⇒bd.

S ⇒ Aa | b

A ⇒ Ac | Aad | bd | ε

Using the algorithm : 2nd iteration of outer loop

- Imposed order of non-terminals : A_1 = S, A_2 = A

```
impose an order on the nonterminals, A1, A2, ..., An

for i ← 1 to n do;
    for j ← 1 to i - 1 do;
        if ∃ a production Ai→Ajγ
            then replace Ai→Ajγ with one or more
                productions that expand Aj
    end;
    rewrite the productions to eliminate
        any direct left recursion on Ai
end;
```

S ⇒ Aa | b

A ⇒ Ac | Aad |

bd | ε

We have to rewrite <u>direct left recursion</u> as <u>right recursion.</u>

Removing direct left recursion in the grammar

```
A_productions = [ A ⇒ Ac , A ⇒ Aad, A ⇒ bd, A ⇒ ε]
Aprime_productions = []
for production in A_productions:
    if production result starts with A:
        move A in the production result to end of result
        popped = A_productions.pop(production)
        replace every A in popped to A'
        A_prime_productions.push(popped)
    else:
        append A' to the production result
Aprime_productions.push(A' ⇒ ε)
```

S ⇒ Aa | b

A ⇒ Ac | Aad |

bd | ε

- **Resulting grammar (right-recursive grammar)**

S ⇒ Aa | b

A ⇒ bdA' | A'

A' ⇒ cA' | adA' | ε

Page 27 of slides from lecture21, covers on how to remove direct left recursion in a grammar.