BLG447 - Derleyici Tasarımı
Dönem Ödevi

Kadir Emre Oto (150140032)

1. Verilen herhangi bir gramere ait kuralları dış ortamdan okuyun ve bellekte uygun bir veri yapısı ile temsil edin. Nonterminal simgeler, simge ve bir işaretçiden oluşan kayıt yapısındaki bir tabloya yerleştirilebilir. Her non-terminalin tanımı olan kural ayrı bir liste yapısı halinde tutulabilir ve tablodaki işaretçi bu listeye işaret eder.

Okuma işlemlerini kolayca yapabilmek ve kodun anlaşılabilirliğini arttırmak için aşağıda belirttiğim 2 farklı class tanımladım. Symbol yapısı gramerdeki herbir karakteri temsil ediyor ve sonradan kullanacağımız değerleri (ilk ve izle kümeleri gibi..) içinde saklıyor. Grammar yapısı ise gramer ile ilgili tüm özellikleri içerisinde barındırıyor.

Grammar yapısındaki Grammar::load(**const char*** filename) metodu dış ortamdan verilen dosyayı okuyup uygun formatta kaydediyor.

Dış ortamdan verilen gramerin şu formatta olduğunu varsayıyorum:

1. % karekteri boş katarı temsil ediyor.

2. Dönüşümlerdeki herbir karakter boşluk ile ayrılmıştır. (Örnek: S -> A B C)

3. Her satırda yalnızca bir dönüşüm verilebilir, farklı satırlarda yeni dönüşümler tanımlayarak ya da operatörünü (|) işlevi yerine getirilebilir.

4. İlk verilen non-terminal karakter start karakteridir.

```
class Symbol{
    friend class Grammar;
    std::stack<Symbol*> S;

    bool is_visited;
    bool is_first_calculated;
    bool is_follow_calculated;
public:
    enum Type {
        nonterminal = 1,
        terminal = 2,
        empty = 3,
        end = 4
    };

    int type;
    const std::string name;

    std::unordered_set<Symbol*> first, follow;
    std::set<std::vector<Symbol*>> transitions;

    Symbol(const char* name);
    Type get_type(const char* name);

    void calculate_first();  // deprecated
    void calculate_follow();  // deprecated
```

```
    void add_transition(std::vector<Symbol*>);
};
     class Grammar {
         Symbol* start;
         std::unordered_map<std::string, Symbol*> symbols;

     public:
         Grammar();
         ~Grammar();

         Symbol* get_symbol(const char* name);
         void add_transition(Symbol* L, std::vector<Symbol*> R);

         void calculate_first_sets();
         void calculate_follow_sets();

         void load(const char* filename);
         void display();
     };
```

İLK ve İZLE kümelerini oluşturun.

İlk ve izle kümeleri oluşturmak için Grammar::calculate_follow_sets() ve Grammar::calculate_first_sets() metotları çağrılmalıdır.

Bu metodları gerçekleştirirken 5. Haftanın ders slaytlarından yararlandım. Programın ürettiği ödev pdf'inde verilen gramerin ilk ve izle kümeleri aşağıdaki gibidir. Grammar::display() metodu gramer ile ilgili önemli bilgileri ekrana yazdırmaktadır.

```
    Bu kümeleri yazmak için yazdığım rekürsiv fonksiyonlar ise bazı durumlarda doğru çalışmadığı için iptal edilmiştir (Symbol::calculate_first(); Symbol::calculate_follow());. Koddaki sıkıntılar düzeltilirse daha verimli çalışabileceği için silinmemiştir.
```

Gramer.txt	Programın çıktısı
S' -> S S -> A B S -> S A B A -> a A -> a a b B -> b B -> b b a	**Start: S'** **$** **First: { }** **Follow: { }** **S** **SAB** **AB** **First: { a }** **Follow: { $ a }** **B** **b** **bba** **First: { b }** **Follow: { $ a }** **A** **a** **aab** **First: { a }** **Follow: { b }** **S'** **S** **First: { a }** **Follow: { $ }**

LR(1) parçalar kümesini (DFA) oluşturun.

Zaman sıkıntıları sebebiyle bu kısmı gerçekleyemedim.

Kodu Derleme ve Çalıştırma

Derleme: g++ kod.cpp Grammar.cpp Symbol.cpp -o kod -std=c++14

Çalıştırma: ./kod

Not: program ile aynı klasörde “gramer.txt” dosyasının olması ve gramerin uygun formatta içinde olması gerekmektedir.