T.C.

SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERLEYİCİ TASARIMINDA SÖZCÜK VE SÖZDİZİM ANALİZİ GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisi : İbrahim AKÇAY

Enstitü Ana Bilim Dalı : Bilişim ve Bilgisayar Mühendisliği

Tez Danışmanı : Yrd. Doç. Dr. Nejat YUMUŞAK

MAYIS 2004

T.C.

SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERLEYİCİ TASARIMINDA SÖZCÜK VE SÖZDİZİM ANALİZİ GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisi : İbrahim AKÇAY

Enstitü Ana Bilim Dalı : Bilişim ve Bilgisayar Mühendisliği

Bu tez 03/06 / 2004 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

.....	.....	.....
Yard. Doç. Dr. Nadir Yumuşak	Yard. Doç. Dr. A. Turhan KELİT	Yard. Doç. Dr. Feyzullah Temurtaş
Jüri Başkanı	**Jüri Üyesi**	**Jüri Üyesi**

TEŞEKKÜR

Bitirme tez konumun seçiminde, çalışmam esnasında bana her zaman önerileri ve konu ile ilgili kaynak konusunda en büyük yardımları sağlayan Bitirme Tez Danışmanım Yrd.Doç.Dr. Nejat YUMUŞAK hocama teşekkürü bir borç bilirim.

Ayrıca her zaman yanımda olan Prof. Dr. Füsun SELÇUK hocamıza ve Akcasu Yazılım Sistemleri Şirketi çalışanlarına teşekkür ederim.

Yüksek Lisans dersleri boyunca bu aşamaya gelmemde benim üzerimde büyük emeği olan Fen Bilimleri Enstitüsü öğretim üyelerine ve çalışanlarına teşekkür ederim.

Her zaman desteklerini arkamda hissettiğim sevgili arkadaşlarım Birkan Sarıfakioğlu, M. Ali Gedek ve Bora Şentürk'e her şey için teşekkür ederim.

Son olarak Lisans ve Yüksek Lisans eğitimim süresince maddi ve manevi desteklerini esirgemeyen, bu seviyeye gelmemde en büyük katkıları olan ve her zaman yanımda olan aileme kucaklar dolusu teşekkürler.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

Simgeler

$\emptyset$: Boş küme

$\cap$: Kesişim

$\cup$: Birleşim

$\in$: Elemanı

$\notin$: Elemanı değil

$\subset$: Alt kümesi

Λ : Boş katar

FA, NFA, NFA - Λ için

L : Dil

M : Makine

Q : Durumlar kümesi

Σ : Giriş sembolünün sonlu alfabesi

q_0 : Başlangıç durumu, $q_0 \in Q$

A : Kabul durumları kümesi, $A \subseteq Q$

δ : $Q \times \Sigma$ 'dan Q'ya geçiş fonksiyonu

CFG için

V : Değişkenler kümesi

Σ : Terminaller

S : Başlangıç sembolü veya değişkeni

P : Üretimler

PDA için

Σ : Giriş Alfabesi

Γ : Yığın Alfabesi

Z_0 : Başlangıç Sembolü

δ : Geçiş Fonksiyonu $Qx(\Sigma \cup \{\Lambda\} x\Gamma \rightarrow (Qx\Gamma^*)$

Durum notasyonları

 Normal Durum

 Kabul Durumu

 Dipsiz Kuyu Durumu

Kısaltmalar

FA : Sonlu Otomatlar (Finite Automata)

NFA : Belirgin olmayan Sonlu Otomatlar (Nondeterministic Finite Automata)

NFA - Λ : Λ Geçişli belirgin olmayan Sonlu Otomatlar (Nondeterministic Finite Automata with Λ Transitions)

CFG : Serbest Bağlamlı Grammer (Contex Free Gramer)

CFL : Serbest Bağlamlı Diller (Contex Free Language)

PDA : Push Down Automata

DPDA : Belirgin PDA (Deterministic Push Down Automata)

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar Kelimeler : Bilgisayar, Derleyici, Sözdizim Analizi, Sözcük Analizi.

Günümüzde derleyiciler daha kapsamlı ve karmaşık yapılara sahiptir. Bu yüzden derleyiciyi oluşturan modüller ve bunların üretimleri konusu gelişim göstermiş ve birçok çözüm yöntemleri geliştirilmiştir.

Genel anlamda bir derleyici altı temel modülden oluşur. Bitirme tezinde bu modüllerden ilk iki aşaması olan Sözcük Analizi ve Sözdizim Analizi konuları ele alınmıştır. Söcük Analizi derleyicinin içerisinde tanımlı dile ait sözcüksel yapıları tanımlar ve bunlar üzerindeki analizi gerçekleştirir. Sözdizim Analizi derleyicinin içerisinde tanımlı dile ait söz dizim yapılarını tanımlar ve analizini gerçekleştirir. Kısaca derleyiciye ait kelime yapılarının analizini Sözcük Analizi cümle yapılarının analizini ise Sözdizim Analizi gerçekleştirmektedir.

Derleyiciye ait her bir modülün elde edilmesi için bir mekanizması ve bu mekanizmanın kullandığı bir otomat tipi vardır. Sözcük Analizi modülü için kullanılan mekanizma Regüler ifadelerdir. Sözcük Analiz içerisinde sözcük yapılarının tanımı Regüler ifadeler olarak tanımlanmaktadır. Dile ait her bir sözcük regüler ifadeler olarak tanımlanır. Sözcük analizini gerçeklemek için Finite Automata kullanılır. Bu amaçla her bir sözcüğe ait NFA diyagram modeli oluşturulur ve girişten gelen bilgi NFA makinesinde analiz edilir.

Sözdizim Analiz modülü için kullanılan mekanizma Context-Free Grammer'dır. Sözdizim Analiz içerisinde söz dizim yapılarının tanımı Context-Free Grammer olarak yapılmaktadır. Dile ait her bir cümle CFG'de tanımlanır. Sözdizim analizini gerçeklemek için Push-Down Automata kullanılır. Bu amaçla oluşturulan CFG'yi PDA'ya dönüştürmek gerekir. PDA'da elde edilen geçiş tablosu ile girişten gelen bilgi PDA tarafından analiz edilir.

IMPLEMENTING SYNTAX AND LEXICAL ANALYSIS IN COMPILER DESIGN

SUMMARY

Keywords : Computer, Compiler, Syntax Analysis, Lexical Analysis.

Today, compilers has more complex and specific structures. Therefore, moduls that creating compilers and its productions has developed with many solutions.

In general, a compiler consist of six basic moduls. These two moduls that are called Lexical Analysis and Syntax Analysis was researched in this study. The Lexical Analysis identify defined expression in compiler language that includes defined statments and analysis under these structures. Breafly, Lexical Analysis process expression of compilers but Syntax Analysis process statments.

There is a mechanism to optain each modul and automata type. It's Regular Expression for Lexical Analysis modul. The definition of statment structures in Lexical Analysis is defined as Regular Expression. Each statment that has a language defined as Regular Expression. Finite Automata is used to analysis word analysis. Therefore, NFA diagram model is created for each word and data that coming from inputs analysis by NFA machine.

The mechanism for Syntax Analysis modul is Context-Free Grammer. The grammers structures in Syntax Analysis defined as Context-Free Grammer. Each sentence that has a language defined in CFG. Push-Down Automata is used to implement grammer analysis. So, it is necessary to replace created CFG to PDA. Inputs and transition table are analysis by PDA.

BÖLÜM 1. GİRİŞ

Bitirme Tez Çalışmasında, Derleyici içerisinde yer alan Sözcük Analizi ve Sözdizim Analizi konuları ayrıntılı olarak incelenmiş ve bu konular kapsamında yer alan modüller ve otomatlar için oluşturulan algoritmaların kullanıldığı bir program geliştirilmesi amaçlanmıştır. Sözcük analizi ve Sözdizim analizi için gerekli bilgiler uygulama programının çalışmasının daha iyi anlaşılması açısından ayrıntılı olarak ele alınmıştır. Program kullanıcı tarafından bir dilin sözcük ve sözdizim yapılarının tanımlanmasını ve yine kullanıcı tarafından verilen bir kaynak program üzerinde bu yapıların analizini sağlamaktadır. Kullanıcı tanımlamak istediği dile ait olan tanımları, sözcük ve sözdizim yapılarını, bunlar arasında ki ilişkileri belirli mekanizmaları kullanarak programa aktarabilir. Bu şekilde bir dile ait tüm kurallar program tarafından tanımlanır ve dile ait herhangi bir kaynak programın sözcük ve sözdizim analizi bu tanımlamalar ve yapılar kullanılarak gerçeklenir.

Programda kullanılan algoritma ve modüller birbirinden bağımsız olarak gerçeklendiğinden istenen modül bağımsız olarak kullanılabilmektedir. Kullanıcı program üzerindeki tüm işlemleri görsel bir ortamda gerçekleştiğinden programa giriş, çıkış ve çalışma anındaki her bir adımda oluşan durumları inceleyebilmektedir. Bu özelliği ile eğitim alanında kullanımı tez kapsamındaki konuların anlaşılması açısından yararlı olabilir.

Derleyici içerisinde yer alan her bir görev, o görevi yerine getirmek için kullanılan bir mekanizmaya ve bu mekanizmayı kullanarak gerekli analizi gerçeklemek için de bir otomata sahiptir.

Bu tez çalışması kapsamında yer alan Sözcük Analizi için kullanılan çalışma mekanizması Regüler Dillerdir. Sözcük Analizi için kullanılan otomat tipi ise Non-

Deterministic Finite Automata'dır. Dilin sözcük yapısı regüler diller kullanılarak tanımlanır. Bir dilin en temel elemanı dilin sembolleridir. Bu semboller birleşerek dil için anlamlı ve geçerli kelimeleri meydana getirir. İşte Sözcük analizi dilin kuralları çerçevesinde tanımlanan bu sözcük yapılarının tanımlanmasını ve bunu kullanarak kaynak program üzerinde sözcüklerin analizini gerçekleştirir. Programda kullanıcı dile ait sözcük yapılarını aşağıdaki yapıları kullanarak tanımlar;

- Programlama diline ait anahtar kelimeler
- Operatörler
- Özel semboller
- Sabitler
- Tipler
- Fonksiyonlar ve tanımlayıcılar.

Burada dile ait her bir sözcük kendi yapısı içerisinde tanımlanır. Bu şekilde Sözcük Analizi sırasında ilgili kelimenin dilin hangi sözcük yapısı içerisinde yer aldığı belirlenir. Bu ilişki Sözdizim Analizi tarafından kelimeler arasında ki ilişkiyi kurmak için kullanılır.

Program Sözcük Analizini gerçeklemek için kullanıcı tarafından verilen kaynak program içerisinde yer alan kelimeleri harf harf alarak yukarıda tanımlanan sözcük yapılarından elde ettiği NFA Diyagramları üzerinden geçerliliğini sınar. Eğer kelime kullanıcı tarafından verilen sözcük yapıları içerisinde oluşturulabiliyorsa dil için anlamlı ve geçerli bir sözcüktür aksi takdirde dil için geçersiz bir sözcüktür.

NDA Diyagramları verilen sözcük yapılarının diyagramının elde edilmesi ve bu diyagram üzerinde kullanıcı tarafından girilen kaynak programa ait sözcüklerin geçerliliğinin testi için kullanılır. Bunun için kullanıcı tarafından verilen sözcük yapıları içerisinde yer alan her bir sözcük için diyagram oluşturulur ve harf harf kontrol edilerek geçerliliği test edilir.

Sözdizim Analizi için kullanılan çalışma mekanizması Context-Free Grammer'dir. CFG dile ait sözdizim yapılarını tanımlar. CFG'de verilen üretim kuralları dile ait

sözcüklerin sözdizim kurallarını tanımlar. Yani dile ait anlamlı ve geçerli sözcüklerin bir araya gelip anlamlı ve geçerli cümle oluşturması için gerekli sıralama yapısını tanımlar. CFG'de verilen üretim kurallarının karmaşıklığına bağlı olarak belirsizlik ortaya çıkmaktadır. Bunun nedeni CFG'de verilen üretim kurallarının aynı anda birden çok üretime izin vermesi ve bu üretimlerin rekürsif bir biçim almasıdır. Bu durumda kullanılan otomat tipinin uyguladığı sözcük analizi işlemi sonuçlanmayabilir yani sonsuz üretim oluşabilir. Bunu ortadan kaldırmak için CFG'de oluşan belirsizliği ortadan kaldırmak gerekir. CFG'de ki belirsizliği ortadan kaldırmanın çeşitli yöntemleri vardır. Ancak belirsizliği ortadan kaldırmak her zaman mümkün değildir.

Sözdizim Analizi otomat olarak Push-Down Automata'yı kullanır. Bunun için kullanıcı tarafından tanımlanan CFG'nin PDA Geçiş Tablosuna dönüştürülmesi gerekmektedir. Bu şekilde verilen üretim kuralları içinde üretim yapısı yığın kullanılarak bir tablo haline dönüştürülür. Elde edilen PDA geçiş tablosu kullanılarak kaynak programda yer alan cümlelerin sözdizim analizi yapılır. CFG'de var olan belirsizlik PDA'da aynı anda birden çok üretim adımına hareket olarak ortaya çıkar.

Bitirme Tez çalışması beş ana bölümden oluşmaktadır.

Bölüm 2'de Sözcük Analizi ve Sözdizim Analizi konularına giriş olması bakımından bir derleyicinin yapısı, modüllerinin tanımlanması ve Chomsky Hiyerarşisi konuları incelenmiştir.

Bölüm 3'de Sözcük Analizi ve Sözdizim Analizi için gerekli olan Diller konusu ayrıntılı olarak incelenmiştir.

Bölüm 4'de Sözcük Analizi ve analizi gerçeklemek için kullanılan FA ve NFA konuları incelenmiştir.

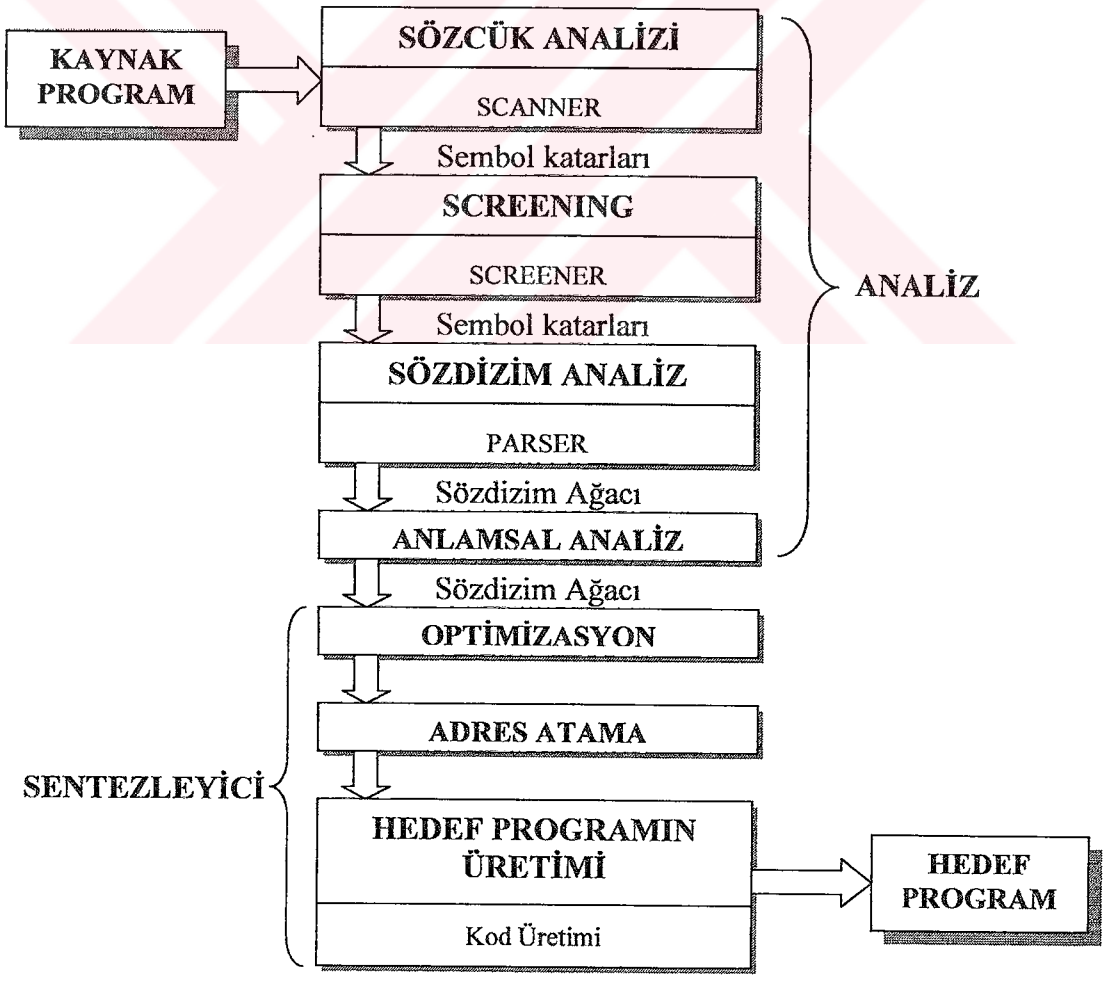
Bölüm 5'de Sözdizim Analizi ve analizi gerçeklemek için gerekli temel bilgiler ve sözdizim yapılarını tanımlamak için CFG ve onun otomatı olan PDA konuları incelenmiştir.

Bölüm 6'de ise yukarıda incelenen konulara ait, yapmış olduğum programın tanıtımı ve uygulamaları anlatılmıştır.

BÖLÜM 2. DERLEYİCİ YAPISI

Derleyicinin en kısa tanımı Kaynak kodu makine koduna çeviren uygulama olarak yapılabilir. Kaynak kod, belirli bir bilgisayar dilinin kurallarına uyularak yazılan koddur. Makine kodu, belirli donanımlar tarafından çalıştırılabilen komutların bulunduğu yapıdır.

Tüm derleyiciler kaynak kodun makine koduna çevrimi aşamasında aşağıdaki adımları izlerler.[1]



Şekil 2.1 Derleyici yapısı

2.1. Sözcük Analizi

Scanner olarak adlandırılan bu modül kaynak programın Sözcük Analiziini gerçekleştirir. Bir scanner kaynak programdan bilgileri karakter karakter okuyarak ilgili programlama dilinin sözcüksel parçalarını elde eder.

Kaynak kod sözcüksel olarak analiz edilir. Örneğin anlatımlar sabitler, tamsayılar, belirleyiciler (Identifiers) şeklinde parçalanır.

Bir programlama dilinin sözcüksel parçaları aşağıdaki gibidir.

- Programlama diline ait anahtar kelimeler
- Operatörler
- Özel semboller
- Sabitler
- Tipler
- Fonksiyonlar ve tanımlayıcılar.

2.2. Screening

Screening Sözcük Analizi tarafından üretilen sembol katarları içerisinde aşağıda verilen sembolleri tanımlamak için kullanılır. Programlama dilinde özel bir anlamı olan semboller. Örnek olarak begin, end, var, int, and vb.
Programlama dili için önemsiz sembolleri elimine eder. Örnek olarak boşluk karakteri açıklama satırları vb.
Programın bir parçası olmayan ancak derleyiciye bazı direktifler veren sembolleri bulur.[1]

2.3. Sözdizim Analizi

Parser olarak adlandırılan Sözdizim Analizi, program sözdizim yapısını tanımlar. Parser programlama dilinin ifade yapısını, deklarasyonlarını ve sabitlerini bilir ve verilen giriş katarı içerisinde programın bu yapılarını tanımlamaya çalışır.[1]

Kaynak kod cümle yapısı bazında kontrol edilir. Örneğin anlatımlar operatörler bazında parçalanır. Parser söz dizim yapısında ki hataları ve yerini belirlemek zorundadır. [1]

2.4. Anlamsal Analiz

Anlamsal Analiz program içerisinde yer alan yapıların anlamsal analizini yapar. Anlamsal Analiz,in yaptığı bazı işlemler aşağıdaki gibi sıralayabiliriz;

- Tip düzeltmelerini yapar,
- Tanımlayıcıların deklare edilmesini kontrol eder.
- Tanımlayıcıların birden fazla deklare edilmesini kontrol eder.

Kaynak kod mantıksal olarak kontrol edilir. Örneğin bir dizi (Array) ile bir tamsayının toplanması bu aşamaya kadar geçerli bir işlemken, bu aşamadan sonra geçersiz sayılır.[1]

2.5. Adres Atama

Sentezleyici adres tahsisi ve ayırması ile başlar. Bu hedef makine için gereklidir. Çünkü makineye ait kelime uzunluğu, adres uzunluğu, direkt olarak adreslenebilen birim makine komutlar vb. işlemler için gereklidir.

2.6. Hedef Programın Üretimi

Code Generator hedef programa ait komutları üretir. Bunun için Kod Üretici bir önceki adımda elde edilen adres atama ve adres değişkenlerini kullanır. Buna rağmen değişken değerleri genellikle makine kaydedicilerinde saklanır. Çünkü buraya erişim genellikle hafızanın herhangi bir yerine erişimden daha hızlıdır. [1]

2.7. Derleyici Modüllerinin Üretimi Ve Biçimsel Gösterimi

Aşağıda bir derleyici içerisinde yer alan temel görevleri yerine getirmek için kullanılan mekanizma ve bunu gerçeklemek için kullanılan otomat tipleri yer almaktadır.[5]

Tablo 2.1 Derleyicinin alt görevleri, çalışma mekanizması ve otomat tipleri

Derleyici görevi	Çalışma mekanizması	Otomat tipi
Sözcük Analizi	Reguler Expression	Deterministic Finite Automata
Sözdizim Analizi	Context-Free Gramer	Deterministic Push-Down Automata
Anlamsal Analiz	Attribute Grammars	
Efficiency-increasing transformations	Tree-Tree transformations	Finite Tree Transductors
Code Selection in Code Generation	Regular Tree Gramer	Finite Tree Automata

Tablo 2.1. de gösterildiği gibi her derleyici görevinin mekanizması ve kullandığını otomat tipi farklıdır. Bunun nedeni derleyici içerisinde farklı görevleri yerine getirebilir.

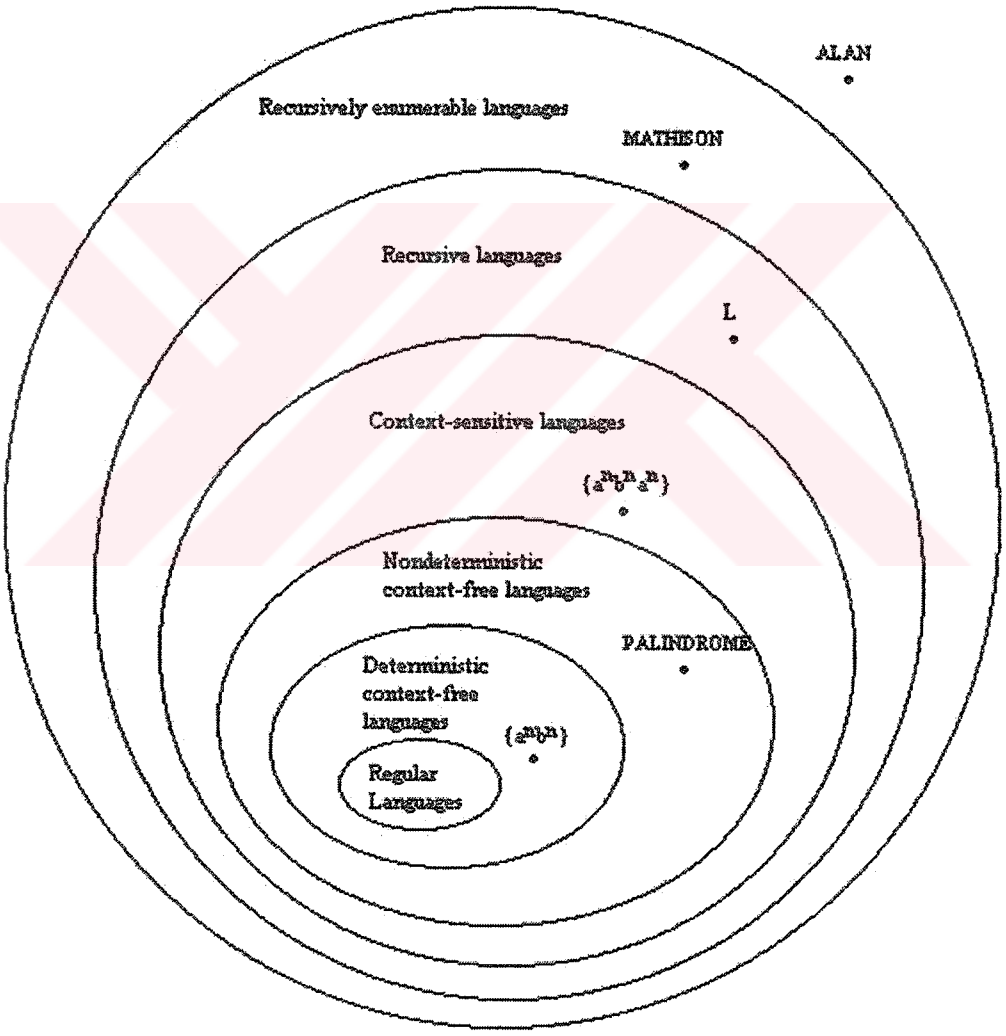
Sözcük Analizi'in derleyici içindeki görevi verilen kaynak kodun sözcüksel olarak analizini gerçekleştirmektir. Bu nedenle verilen kaynak kodda tanımlı olan dile ait sözcüksel parçaları arar. Sözcük Analizi modülü, dilin sözcüksel yapılarının Regüler Expression olarak tanımlanması ve bu ifadelerin NFA diyagramlarının elde edilmesi ile gerçeklenebilir.

Yine aynı şekilde Sözdizim Analiz modülü, dilin sözdizim yapılarını tanımlamak için kullanıldığından dile ait bu sözdizim yapılarının CFG ifadelerinin oluşturulması ve bu ifadelerin PDA makinesine dönüştürülmesi ile gerçeklenebilir.

2.8. Chomsky Hiyerarşisi

Regüler ve diğer diller Noam Chomsky tarafından aşağıdaki gibi sınıflandırılmıştır.

Gramer	Otomat Tipi
Regular	NFA or DFA
Context-Free	Push-Down Automaton
Context-Sensitive	Linear-Bounded Automaton
Unrestricted (or *Free*)	Turing Machine



Şekil 2.2. Dillerin sınıflandırılması

BÖLÜM 3. DİLLER

TANIM : Bir alfabeden alınan sembollerden oluşmuş katarlar kümesidir. L sembolü ile gösterilir.[2]

3.1. Alfabe

Belli sembollerden oluşmuş katarlardır. Σ sembolü ile gösterilir.

Σ = { a , b } a, ab, aabb, bbba

Alfabe Semboller Dilin Sözcükleri

Λ = NULL STRING = Boş katar.

Σ* : Σ alfabesinden türetilen olası tüm katarlar kümesidir.

3.2.Dillerin tanımlanması

Diller bir alfabeden alınan sembollerden oluşan kümelerdir.

$\Sigma = \{ x \}$

$L_1 = \{x, xx, xxx, xxxx, ...\}$

$L_1 = \{x^n \text{ için } n = 1,2,3,....\}$

3.3. Katar uzunluğu

TANIM : x bir katar ise x katarının uzunluğu $length(x) = | x |$ ile gösterilir.

ÖRNEK: $x = \Lambda$ $length(x) = | x | = 0$

x = 428 $length(x) = |x| = 3$

x = abba $length(x) = |x| = 4$

3.4. Katar tersi

TANIM : x Σ alfabesinden üretilen L_1 diline ait bir katar ise x katarının tersi $reverse(x) = x^r$ ile gösterilir.

ÖRNEK : x = aaa $reverse(x) = aaa$

x = 412 $reverse(x) = 214$

3.5. Polindrom

TANIM : Bir Σ alfabesinde tanımlanan L_1 diline ait bir x katarı $Polindrom = \{\Lambda$, ve tüm x'ler $reverse(x) = x\}$ sağlıyorsa x katarı POLİNDROM' dur denir.

ÖRNEK : $\Sigma = \{a, b\}$ $Polindrom = \{\Lambda, a, b, aa, bb, aaa, aba, bbb, bab, ...\}$

3.6. Kleene* teoremi

Kleene* işlemi bir ifade de istenilen kadar kullanımı sağlar.

ÖRNEK : $\Sigma = \{x\}$ ise $\Sigma^* = \{\Lambda, x, xx, xxx, xxxx\}$

$\Sigma = \{0, 1\}$ ise $\Sigma^* = \{\Lambda, 0, 1, 00, 01, ...\}$

TANIM : S bir katar kümesi ise S*'da bu kümeden istenilen kadar alınarak elde edilen kümedir.

ÖRNEK : $S = \{aa.b\}$ ise $S^* = \{\Lambda, aa, b, aab, baa, aabaa\}$

aabaa = (aa) (b) (aa)

$xxxxx = (xx)(xxx)$ yazılabilir.

$\Sigma = \{x()\}$ $\quad$ *length*(xxxxx)= 5

length((*xx*)(*xxx*)) = 9

Bazı Gösterimler ve Örnekler :

Eğer $L_1, L_2 \subseteq \Sigma^*$ ise

$L_1 L_2 = \{xy \mid x \in L_1 \text{ ve } y \in L_2\}$

$L' = \Sigma^* - L$

Eğer $a \in \Sigma, x \in \Sigma^*, L \subseteq \Sigma^*$ ise

$a^k = aa.....a$ $\quad$ $x^k = xx.....x$

$\Sigma^k = \Sigma\Sigma.....\Sigma$ $\quad$ $L^k = LL.....L$

$a^0 = \Lambda$ $\quad$ $x^0 = \Lambda$

$\Sigma^0 = \{\Lambda\}$ $\quad$ $L^0 = \{\Lambda\}$

$L^* = \bigcup_{i=0}^{\infty} L^k$ $\quad$ $L^+ = \bigcup_{i=1}^{\infty} L^k$

$L' = \Sigma^* - L$

3.7. Ekleme

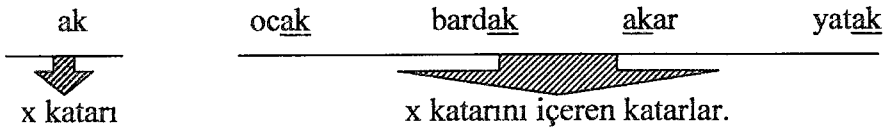
TANIM : $x, y \in \Sigma^*$ ise xy'e x ve y katarının eklemesi denir.

$x = 001$, $y = 11$ $\Rightarrow$ $xy = 00111$, $yx = 11001$ $\Rightarrow$ $xy \neq yx$

$x\Lambda = \Lambda x = x$

$x, y, z \in \Sigma^*$ tanımlı katarlar ise $x(yz) = (xy)z$

Bir y, w ve z $y = wxz$ biçiminde yazılabiliyorsa x y'nin bir alt katarıdır.



$L^* = \bigcup_{k=0}^{\infty} L^k$ KLEENE* işlemi.

$L^+ = \bigcup_{k=1}^{\infty} L^k = L\ L^*$ bir veya daha çok sayıda elemanların eklenmesi ile oluşan küme.

Örnekler :

$L = \{0,11\}^* \cup \{1\}\{11\}^+$

$\{0,11\}^*$: 0 veya 11 elemanlarından istenilen sayıda alınarak oluşturulabilen katarlar.

$\{1\}$: '1' elemanı zorunlu olarak kullanılmalıdır. L dilinin sözcüklerinde bu eleman bulunmalıdır.

$\{11\}^+$: 11 elemanından en az bir kere alınarak oluşturulabilen katarlar.

L diline ait örnek katarlar

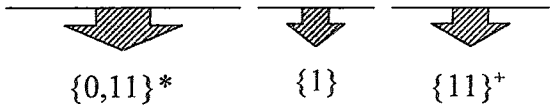
x = 1 11

x = 0 1 11

x = 11 1 11

x = 0 11 1 11

x = 0 11 0 11 0 11 1 11 11 11



L diline ait olmayan örnek katarlar

x = 11

x = 0 1

x = 0 11

x = 0 11 11

x = 11 11 11 11 11

3.8. Regüler diller ve regüler ifadeler

Bir Σ alfabesinden bir "REGÜLER DİL" 0 veya 1 uzunluklu tek bir katar içeren s üzerinden bir dilden sadece BİRLEŞİM, EKLEME ve KLEENE* işlemleri kullanılarak elde edilen dildir. "REGÜLER İFADE" ise ifadedeki {} yerine () veya hiçbir şey, ∪ yerine + konulması ile elde edilir. [2]

Örnek :

Dil	Regüler İfade
$\{\Lambda\}$	Λ
$\{0\}$	0
$\{0,1\} \equiv (\{0\} \cup \{1\})$	$0+1$
$\{0,\Lambda\}\{0,11\} \equiv (\{0\} \cup \{\Lambda\}\{0\} \cup \{11\})$	$(0+\Lambda)(0+11)$
$\{0,1\}^*\{11,01\}^+$	$(0+1)^*(11+01)^+$

TEOREM : Σ üzerine R regüler diller sınıfı ve karşılık düşen regüler ifadeler aşağıdaki gibi tanımlanır.

ø, R'nin bir elemanıdır. Karşılık düşen regüler ifade ø' dır.

{Λ}, R'nin bir elemanıdır. Karşılık düşen regüler ifade Λ ' dır.

Her $a \in \Sigma$ için { a } R'nin elemanıdır. Karşılık düşen regüler ifade a' dır.

L_1 ve L_2 R'nin elemanıdır. Karşılık düşen regüler ifade ise

$L_1 \cup L_2$ R'nin elemanıdır. Karşılık düşen regüler ifade $r_1 + r_2$

L_1 L_2 R'nin elemanıdır. Karşılık düşen regüler ifade r_1 r_2

L_1 * R'nin elemanıdır. Karşılık düşen regüler ifade r_1 *

BÖLÜM 4. SÖZCÜK ANALİZİ

4.1. Finite Automata – FA

TANIM : Bir sonlu otomat (Finite Automata – FA) yada sonlu durumlu makine (Finite State Machine) bir $M = \{Q, \Sigma, q_0, A, \delta\}$ beşlisinden oluşur. [2]

M : Makine

Q : Durumlar kümesi

Σ : Giriş sembolünün sonlu alfabesi

q_0 : Başlangıç durumu, $q_0 \in Q$

A : Kabul durumları kümesi, $A \subseteq Q$

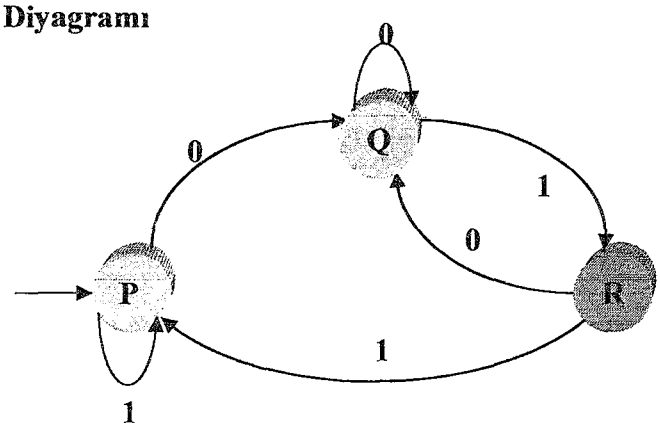
δ : $Q \times \Sigma$ 'dan Q'ya geçiş fonksiyonu

4.1.1. Özellikleri

Tüm geçişler olmak zorunda

Aynı geçişten birden fazla sayıda olamaz.

4.1.2. FA Diyagramı



Şekil 4.1. FA Diyagramı

$M = \{Q, \Sigma, q_0, A, \delta\}$

$\Sigma = \{0,1\}$

$Q = \{P, Q, R\}$

$q_0 = \{P\}$

$A = \{R\}$

4.1.3. FA Diyagramın Geçiş Fonksiyonları

$\delta^*(q,y) = \delta\ (P,0) = \{Q\}$ $\quad$ $\delta^*(q,y) = \delta\ (P,1) = \{P\}$

$\delta^*(q,y) = \delta\ (Q,0) = \{Q\}$ $\quad$ $\delta^*(q,y) = \delta\ (Q,1) = \{R\}$

$\delta^*(q,y) = \delta\ (R,0) = \{Q\}$ $\quad$ $\delta^*(q,y) = \delta\ (R,1) = \{P\}$

Tablo 4.1. FA Diyagramın Geçiş Fonksiyonları

q	$\delta(q,0)$	$\delta(q,1)$
P	Q	P
Q	Q	R
R	Q	P

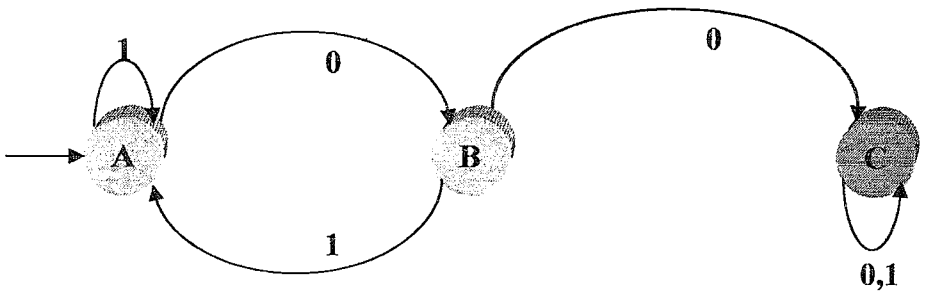
4.1.4. FA için δ^*'in rekürsif tanımı

TANIM :

Herhangi bir $q \in Q$ için $\delta^*(q,\Lambda) = q$

Herhangi bir $y \in \Sigma^*, a \in \Sigma$ ve $q \in Q$ için $\delta^*(q,ya) = \delta(\delta^*(q,y),a)$

4.1.5. FA'da Verilen katarın Test Etme



Şekil 4.2. FA'da verilen bir katarı test etme

$M=\{Q,\Sigma,q_0,A,\delta\}$

$\Sigma=\{0,1\}$

$Q=\{A,B,C\}$

$q_0=\{A\}$

$A=\{C\}$

Diyagramın Geçiş Fonksiyonları

$\delta^*(q,y)=\delta\ (A,0)=\{B\}$ $\quad$ $\delta^*(q,y)=\delta\ (A,1)=\{A\}$

$\delta^*(q,y)=\delta\ (B,0)=\{C\}$ $\quad$ $\delta^*(q,y)=\delta\ (B,1)=\{A\}$

$\delta^*(q,y)=\delta\ (C,0)=\{C\}$ $\quad$ $\delta^*(q,y)=\delta\ (C,1)=\{C\}$

Tablo 4.2. FA Diyagramın Geçiş Fonksiyonları

q	$\delta(q,0)$	$\delta(q,1)$
A	B	A
B	C	A
C	C	C

x = 0101 olsun.

Geçiş fonksiyonu

$\delta^*(q,ya)=\delta(\delta^*(q,y),a)$ ise

$\delta^*(A,0101) = \delta(\delta^*(A,010),1)$

$= \delta(\delta(\delta^*(A,01),0),1)$

$= \delta(\delta(\delta(\delta^*(A,0),1),0),1)$

$= \delta(\delta(\delta(\delta^*(A,\Lambda 0),1),0),1)$

$= \delta(\delta(\delta(\delta(\delta^*(A,\Lambda),0),1),0),1)$

$= \delta(\delta(\delta(\delta^*(A,0),1),0),1)$

$= \delta(\delta(\delta^*(B,1),0),1)$

$= \delta(\delta^*(A,0),1)$

$= \delta(B,1)$

$= \{ A \}$ x = 0101 katarı RED edildi.

4.1.6. Fa kabul durumu

TANIM : $M = \{Q, \Sigma, q_0, A, \delta\}$ bir FA olsun. Bir $x \in \Sigma^*$ katarı eğer $\delta^*(q_0, x) \in A$ ise M Makinesi tarafından kabul edilmiş olur.

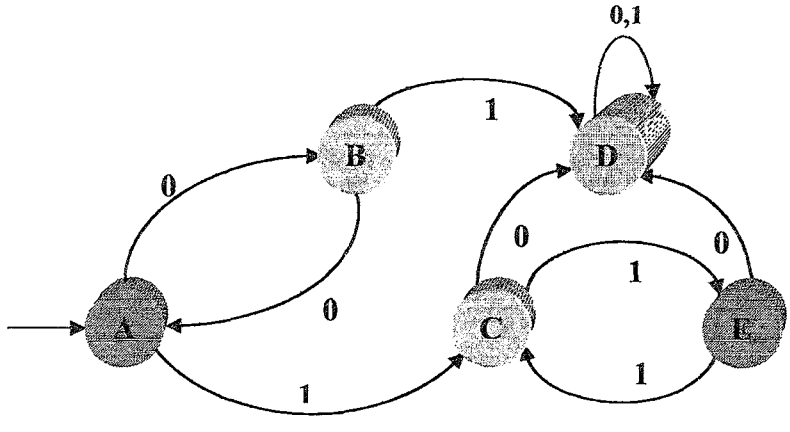
L (M) : M tarafından kabul edilen dil.

$L(M) = \{ x \in \Sigma^* | x,$ Makinesi tarafından kabul edilmiştir. $\}$

L , Σ üzerinden tanımlanmış bir dil ise L dili M tarafından yalnız ve yalnız L = L (M) ise kabul edilir.

TEOREM : Bir Σ alfabesi üzerine bir L dili yalnız ve yalnız L'yi kabul eden bir FA varsa regülerdir.

ÖRNEK :



Şekil 4.3. Örnek FA Diyagramı

FA Makine : $M=\{Q,\Sigma,q_0,A,\delta\}$

Alfabe : $\Sigma=\{0,1\}$

Durumlar Kümesi : $Q=\{A,B,C,D,E\}$

Başlangıç Durumu : $q_0=\{A\}$

Kabul Durumları K. : $A=\{A,E\}$

Kabul durumlarına ulaşmak için gereken geçişleri bulalım.

A'ya ulaşmak için (00)*

A'dan E'ye ulaşmak için (00)* 11 (11)*

Birleştirirsek $(00)^*\,11(11)^* + (00)^* = (00)^*\,(11)^+$ elde ederiz.

4.1.7. Birleşim, ekleme ve kleene* işlemleri

Eğer $M_1=\{Q_1,\Sigma,q_1,A_1,\delta_1\}$ ve $M_2=\{Q_2,\Sigma,q_2,A_2,\delta_2\}$ makinelerine varsa ve L_1 ve L_2 bu makinelerin kabul ettiği diller ise

$Q_1 \times Q_2$ olası durumlar

$q_0=(q_1,q_2)$ Başlangıç durumu

$p\in Q_1$, $q\in Q_2$ ise $\delta(\delta_1(p,a),\delta_2(q,a))$ Geçiş fonksiyonu elde edilir.

TEOREM : $M_1=\{Q_1,\Sigma,q_1,A_1,\delta_1\}$ ve $M_2=\{Q_2,\Sigma,q_2,A_2,\delta_2\}$ L_1 ve L_2 dillerini kabul eden FA'lar olsun. M ise

$M=\{Q,\Sigma,q_0,A,\delta\}$

$Q = Q_1 \times Q_2$ Durumlar

$q_0=(q_1,q_2)$ Başlangıç durumu

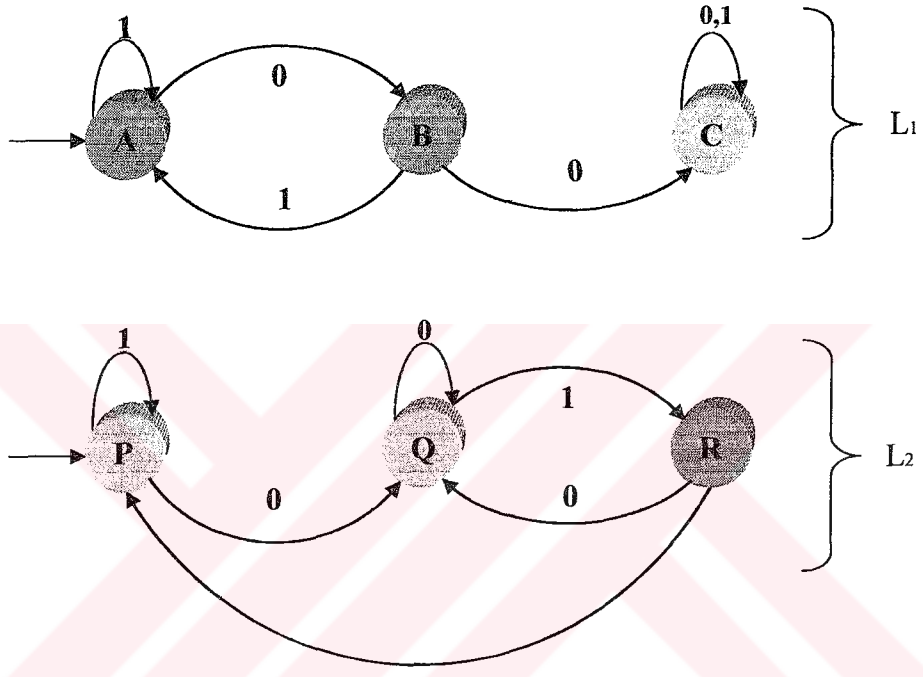
$\delta(\delta_1(p,a),\delta_2(q,a))$, $p\in Q_1$, $q\in Q_2$, $a\in\Sigma$ Geçiş Fonksiyonu olarak tanımlansın.

Eğer $A=\{(p,q)|p\in A_1$ VEYA $q\in A_2\}$ ise $M, L_1 \cup L_2$ 'yi kabul eder.

Eğer $A=\{(p,q)|p\in A_1$ VE $q\in A_2\}$ ise $M, L_1 \cap L_2$ 'yi kabul eder.

Eğer $A=\{(p,q)|p\in A_1$ VE $q\notin A_2\}$ ise $M, L_1 - L_2$ 'yi kabul eder.

ÖRNEK : $L_1=\{x|x$ 00 alt katarı içermez$\}$ $L_2=\{x|x$ 01 ile biter$\}$



Şekil 4.4. Verilen iki FA Diyagramın Kabul Durumları

$Q_1=(A,B,C)$

$Q_2=(P,Q,R)$

$Q=Q1 \times Q2=\{(A,P),(A,Q),(A,R),(B,P),(B,Q),(B,R),(C,P),(C,Q),(C,R)\}$

$q_0=(q_1,q_2)=(A,P)$

$A_1=\{A,B\}$

$A_2=\{R\}$

Şimdi diyagramın FA Tablosu oluşturulabilir. Diyagramlar başlangıç durumlarından başlanarak geçişler ortak yapılır ve geçiş sonucunda ulaşılan durumlar yeni durumlar olarak alınır ve tüm geçişlerin incelenmesi ile FA Tablosu elde edilir. Bu tablo iki

FA diyagramının birleştirilmesidir. Artık bu FA üzerinde $L_1 \cup L_2$ $L_1 \cap L_2$ ve $L_1 - L_2$ işlemleri gerçekleştirilerek yeni kabul durumları elde edilir.

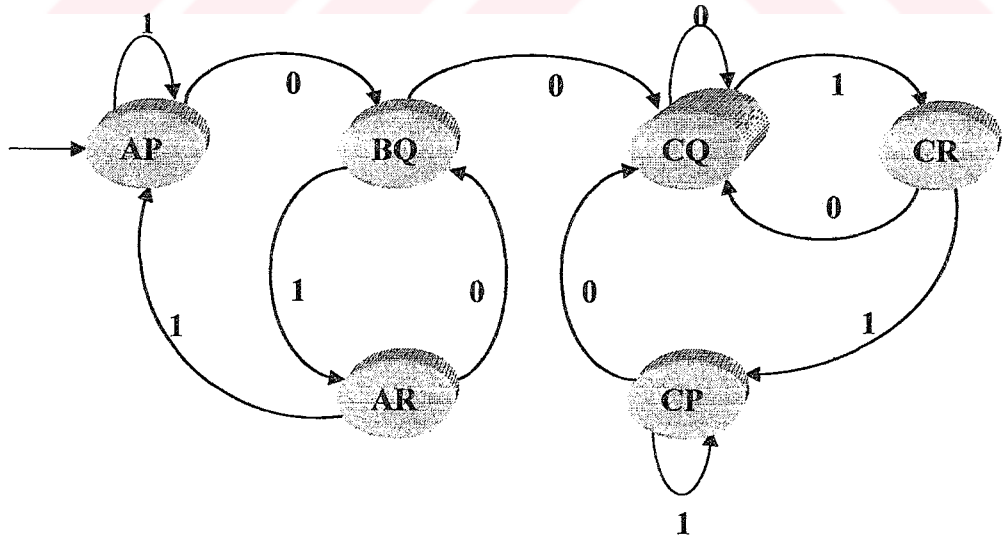
Tablo 4.3. FA Geçiş Tablosu

q	$\delta(q,0)$	$\delta(q,1)$
{A,P}	{A,P}	{B,Q}
{B,Q}	{A.R}	{C,Q}
{C,Q}	{C,R}	{C,Q}
{A.R}	{A,P}	{B,Q}
{C,R}	{C,P}	{C,Q}
{C,P}	{C,P}	{C,Q}

$L_1 \cup L_2 \Rightarrow \quad A = \{(A,P),(A,R),(B,Q),(C,R)\}$

$L_1 \cap L_2 \Rightarrow \quad A = \{(A,R)\}$

$L_1 - L_2 \Rightarrow \quad A = \{(A,P),(B,Q)\}$



Şekil 4.5. Birleşmiş FA Diyagramı

4.2. Nondeterministic Finite Automata–NFA

TANIM : Bir NonDeterministic Finite Automata – NFA $M = \{Q, \Sigma, q_0, A, \delta\}$ beşlisi olup;

M : Makine

Q : Durumlar kümesi

Σ : Giriş sembolünün sonlu alfabesi

q_0 : Başlangıç durumu, $q_0 \in Q$

A : Kabul durumları kümesi, $A \subseteq Q$

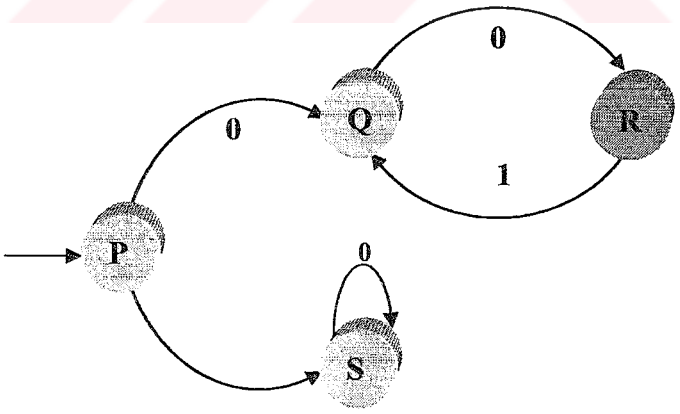
δ : Q x Σ'dan Q'ya geçiş fonksiyonu

4.2.1. Özellikleri

Bazı girişler olmayabilir.

Aynı geçişten birden fazla sayıda olabilir.

4.2.2. NFA diyagramı



Şekil 4.6. NFA Diyagramı

$M = \{Q, \Sigma, q_0, A, \delta\}$

$\Sigma = \{0, 1\}$

$Q = \{P, Q, R, S\}$

$q_0 = \{P\}$

$A = \{R\}$

4.3.3. NFA geçiş tablosu

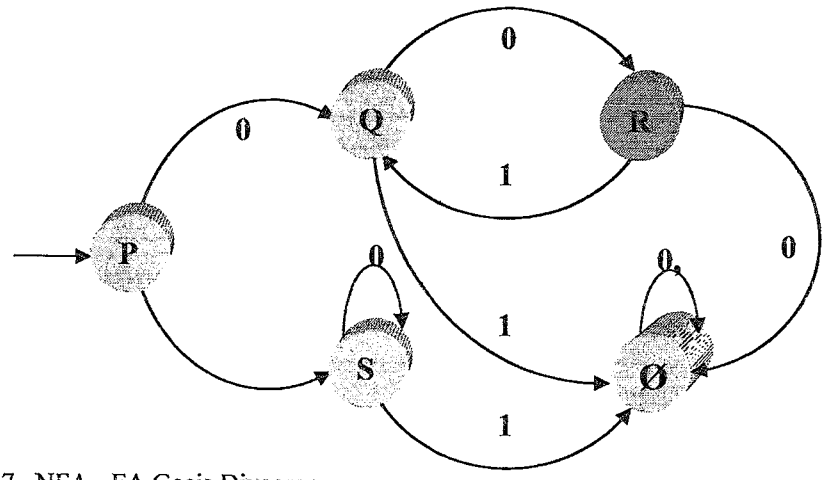
Tablo 4.4. NFA Geçiş Tablosu

q	$\delta(q,0)$	$\delta(q,1)$
P	Q	S
S	S	Ø
Q	R	Ø
R	Ø	Q

4.2.4. NFA diyagramının FA geçiş tablosu

Tablo 4.5. NFA – FA Geçiş Tablosu

q	$\delta(q,0)$	$\delta(q,1)$
P	Q	S
S	S	Ø
Q	R	Ø
Ø	Ø	Ø
R	Ø	Q



Şekil 4.7. NFA - FA Geçiş Diyagramı

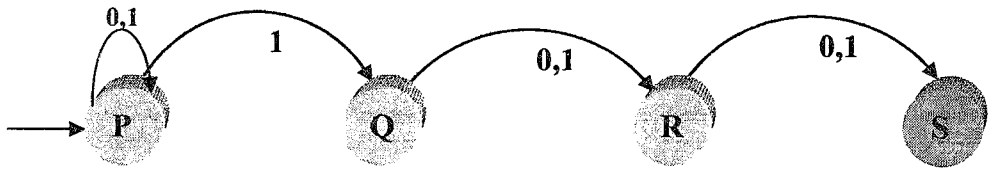
4.2.5. NFA için δ^*,in rekürsif tanımı

TANIM :

Herhangi bir $q \in Q$ için $\delta^*(q, \Lambda) = \{q\}$

Herhangi bir $y \in \Sigma^*, a \in \Sigma$ ve $q \in Q$ için $\delta^*(q, ya) = \bigcup_{p \in \delta^*(q,y)} \delta(p, a)$

4.2.6. NFA' da verilen bir katarın test işlemi



Şekil 4.8. NFA' da verilen bir katarı test etme

$M = \{Q, \Sigma, q_0, A, \delta\}$

$\Sigma = \{0, 1\}$

$Q = \{P, Q, R, S\}$

$q_0 = \{P\}$

$A = \{S\}$

NFA Geçiş Tablosu

Tablo 4.6. NFA Geçiş Tablosu

q	$\delta(q,0)$	$\delta(q,1)$
P	P	P,Q
Q	R	R
R	S	S
S	Ø	Ø

İncelenecek katar x = 111 olsun.

$\delta^*(P,111)=\bigcup_{p\in\delta^*(P,11)}\delta(p,1)$

$= \delta(P,1)\cup\delta(Q,1)\cup\delta(R,1)$

$= \{P,Q,R,S\}$ S Kabul durumu küme içinde olduğundan x katarı KABUL edildi. İncelenecek katar x = 01 olsun.

$\delta^*(P,111)=\bigcup_{p\in\delta^*(P,0)}\delta(p,1)$

$= \delta(P,1)$

$= \{P,Q\}$ Kabul durumu küme içinde olmadığından x katarı RED edildi.

4.2.7. NFA kabul durumu

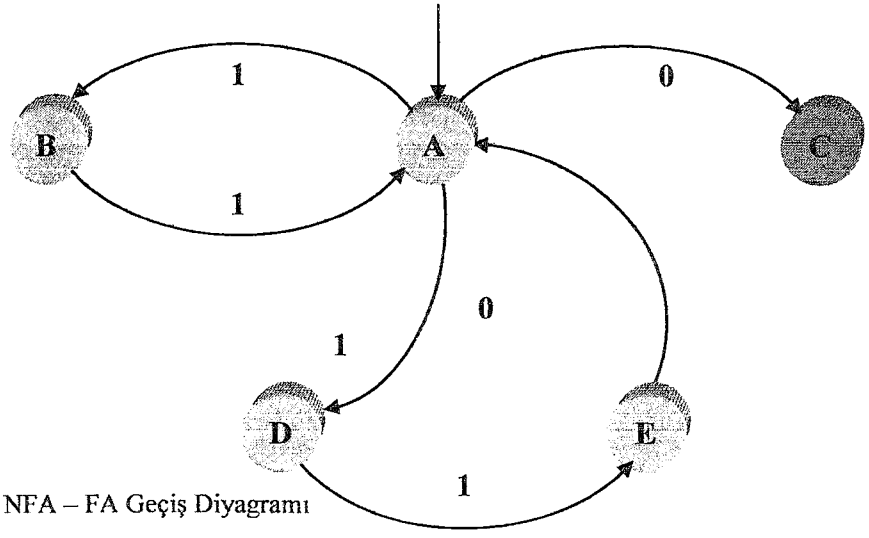
TANIM : $M=\{Q,\Sigma,q_0,A,\delta\}$ bir NFA olsun. Bir $x\in\Sigma^*$ katarı eğer $\delta^*(q_0,x)\cap A\neq\phi$ ise M makinesi tarafından kabul edilmiş olur.

L (M) : M tarafından kabul edilen dil.

$L(M)=\{x\in\Sigma^*|$ x, Makinesi tarafından kabul edilmiştir. $\}$

L , Σ üzerinden tanımlanmış bir dil ise L dili M tarafından yalnız ve yalnız L = L (M) ise kabul edilir.

ÖRNEK 2 : NFA'dan FA Geçiş Tablosu



Şekil 4.9. NFA – FA Geçiş Diyagramı

$M = \{Q, \Sigma, q_0, A, \delta\}$

$\Sigma = \{0,1\}$

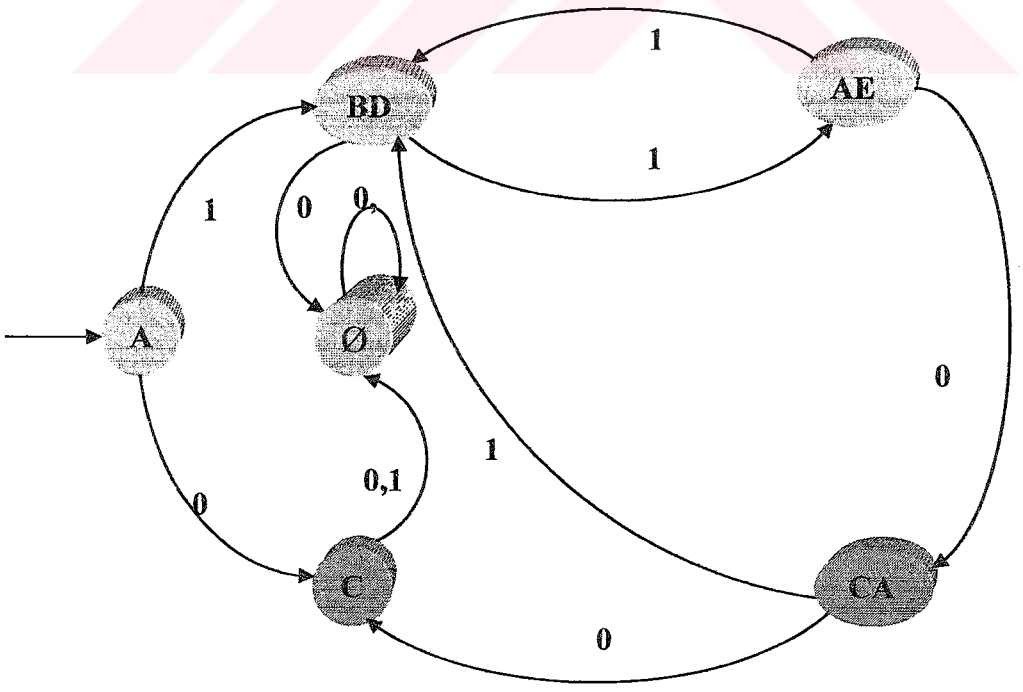
$Q = \{A, B, C, D, E\}$

$q_0 = \{A\}$

$A = \{C\}$

Tablo 4.7. NFA – FA Geçiş Tablosu

q	$\delta(q,0)$	$\delta(q,1)$
A	C	B,D
C	Ø	Ø
B,D	Ø	A,E
Ø	Ø	Ø
A,E	A,C	B,D
A,C	C	B,D



Şekil 4.10. FA Diyagramı

4.3. Λ Geçişli NFA (NFA-Λ)

TANIM : Bir Λ geçişli NFA (NFA - Λ) bir $M=\{Q,\Sigma,q_0,A,\delta\}$ beşlisi olup

M : Makine
Q : Durumlar kümesi
Σ : Giriş sembolünün sonlu alfabesi
q_0 : Başlangıç durumu, $q_0 \in Q$
A : Kabul durumları kümesi, $A \subseteq Q$
δ : Q x Σ 'dan Q'ya geçiş fonksiyonu

4.3.1. Özellikleri

Bazı girişler olmayabilir.
Aynı geçişten birden fazla sayıda olabilir.
En az bir Λ geçişi olmalıdır.

4.3.2. Durumların Λ Kapanması

TANIM : Bir durumlar kümesinin Λ Kapanması

$M=\{Q,\Sigma,q_0,A,\delta\}$ bir NFA - Λ makinesi olsun; S Q'nun herhangi bir alt kümesi olsun. S kümesinin Λ Kapanması ola $\Lambda(S)$;

S'in her elemanı, $\Lambda(S)$'in elemanıdır.
Herhangi için $\delta(q,\Lambda)$'in her elemanı $\Lambda(S)$ de vardır.
Başka bir yolla (1 ve 2 dışında) elde edilecek hiçbir Q elemanı $\Lambda(S)$'e ait değildir.

4.3.3. Λ (S) Algoritması

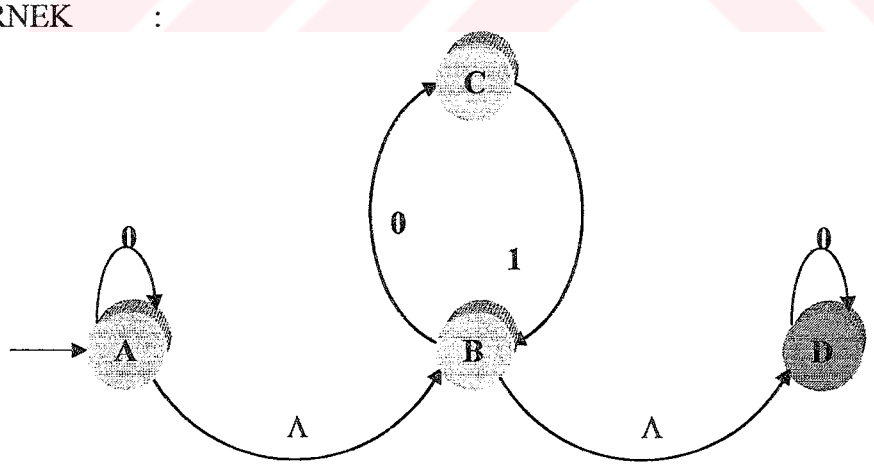
TANIM : T = S ile başla, her adımda $q \in T$ elemanları için tüm $\delta(q,\Lambda)$ kümelerini ele al ve bu kümelerin herhangi birinde olursa olsun herhangi bir elemanı o ana kadar T' de değilse ekle. Bu işlemi T değişmez olana kadar sürdür. Λ(S) kümesi T'nin son değeridir.

$\delta^*(q,y)$: q durumunda y sembollerini ve Λ geçişlerini de kullanarak erişilen tüm durumların kümesi ise ;

$\bigcup_{p \in \delta^*(P,y)} \delta(p,a)$: kümesi q sembolünü kullanarak erişebileceğimiz tüm durumlar kümesidir.

Bu kümenin Λ kapanması ise ek olarak Λ Geçişlerine izin verdiğimiz zaman erişebileceğimiz diğer tüm durumları verir.

ÖRNEK :



Şekil 4.11. NFA - Λ Diyagramı

4.3.4. NFA - Λ Tablosu

Tablo 4.8. NFA - Λ Geçiş Tablosu

q	$\delta(q,\Lambda)$	$\delta(q,0)$	$\delta(q,1)$
A	B	A	Ø
B	D	C	Ø
C	Ø	Ø	B
D	Ø	D	Ø

Tablo 4.9. Durumların Λ Kapanması

Q	Λ(q)
A	A , B , D
B	B , D
C	C
D	D

4.3.5. NFA - Λ 'dan NFA' ya Geçiş Tablosu

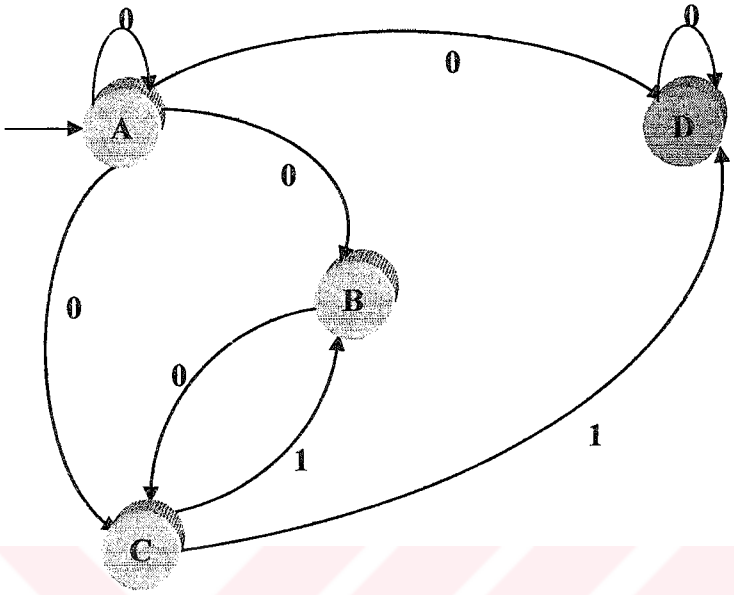
Tablo 4.10. NFA Λ - NFA' ya Geçiş Tablosu

q	$\delta(q,\Lambda)$	$\delta(q,0)$	$\delta(q,1)$	$\delta^*(q,0)$	$\delta^*(q,1)$
A	B	A	Ø	A,B,C,D	Ø
B	D	C	Ø	C,D	Ø
C	Ø	Ø	B	Ø	B,D
D	Ø	D	Ø	D	Ø

NFA Λ Tablosu

NFA Geçiş Tablosu

NFA Diyagramın Elde Edilmesi



Şekil 4.12. NFA Diyagramı

4.3.6. NFA - Λ Kabul Durumu

TANIM : $M=\{Q,\Sigma,q_0,A,\delta\}$ bir NFA - Λ olsun. Bir $x\in\Sigma^*$ katarı eğer $\delta^*(q_0,x)\cap A\neq\phi$ ise M makinesi tarafından kabul edilmiş olur.

L (M) : M tarafından kabul edilen dil.

$L(M)=\{x\in\Sigma^*|\, x,$ Makinesi tarafından kabul edilmiştir. $\}$

L , Σ üzerinden tanımlanmış bir dil ise L dili M tarafından yalnız ve yalnız L = L (M) ise kabul edilir.

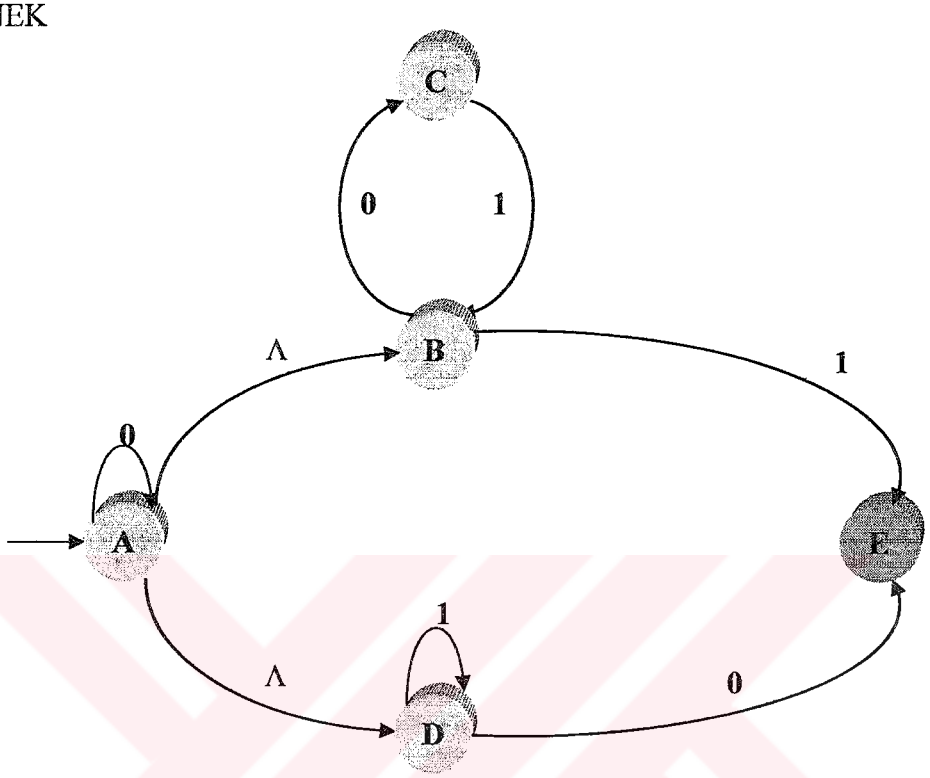
4.3.7. NFA - Λ için δ^*'in rekürsif tanımı

TANIM :

Herhangi bir $q\in Q$ için $\delta^*(q,\Lambda)=\Lambda(\{q\})$

Herhangi bir $y \in \Sigma^*, a \in \Sigma$ ve $q \in Q$ için $\delta^*(q, ya) = \Lambda\left(\bigcup_{p \in \delta^*(q,y)} \delta(p,a)\right)$

ÖRNEK :



Şekil 4.13. NFA Λ Diyagramı

NFA - Λ Tablosu

Tablo 4.11. NFA Λ Geçiş Tablosu

q	$\delta(q,\Lambda)$	$\delta(q,0)$	$\delta(q,1)$
A	B,D	A	Ø
B	Ø	C	E
C	Ø	Ø	B
D	Ø	E	D
E	Ø	Ø	Ø

Durumların Λ Kapanması

Tablo 4.12. Durumların Λ Kapanması

q	Λ(q)
A	A , B , D
B	B
C	C
D	D
E	E

İncelenecek katar x = 011 olsun.

$$\delta^*(A,\Lambda)=\Lambda(\{A\})=\{A,B,D\}$$

$$\delta^*(A,0)=\Lambda\left(\bigcup_{p\in\delta^*(A,\Lambda)}\delta(p,0)\right)$$

$$=\Lambda(\delta(A,0)\cup\delta(B,0)\cup\delta(D,0))$$

$$=\Lambda(\{A\}\cup\{C\}\cup\{E\})$$

$$=\Lambda(\{A,C,E\})$$

$$=\{A,B,C,D,E\}$$

$$\delta^*(A,01)=\Lambda\left(\bigcup_{p\in\delta^*(A,0)}\delta(p,1)\right)$$

$$=\Lambda(\delta(A,1)\cup\delta(B,1)\cup\delta(C,1)\cup\delta(D,1)\cup\delta(E,1))$$

$$=\Lambda(\phi\cup\{B\}\cup\phi\cup\{D\}\cup\delta\{E\})$$

$$=\Lambda(\{B,D,E\})$$

$$=\{B,D,E\}$$

$$\delta^*(A,011)=\Lambda\left(\bigcup_{p\in\delta^*(A,1)}\delta(p,1)\right)$$

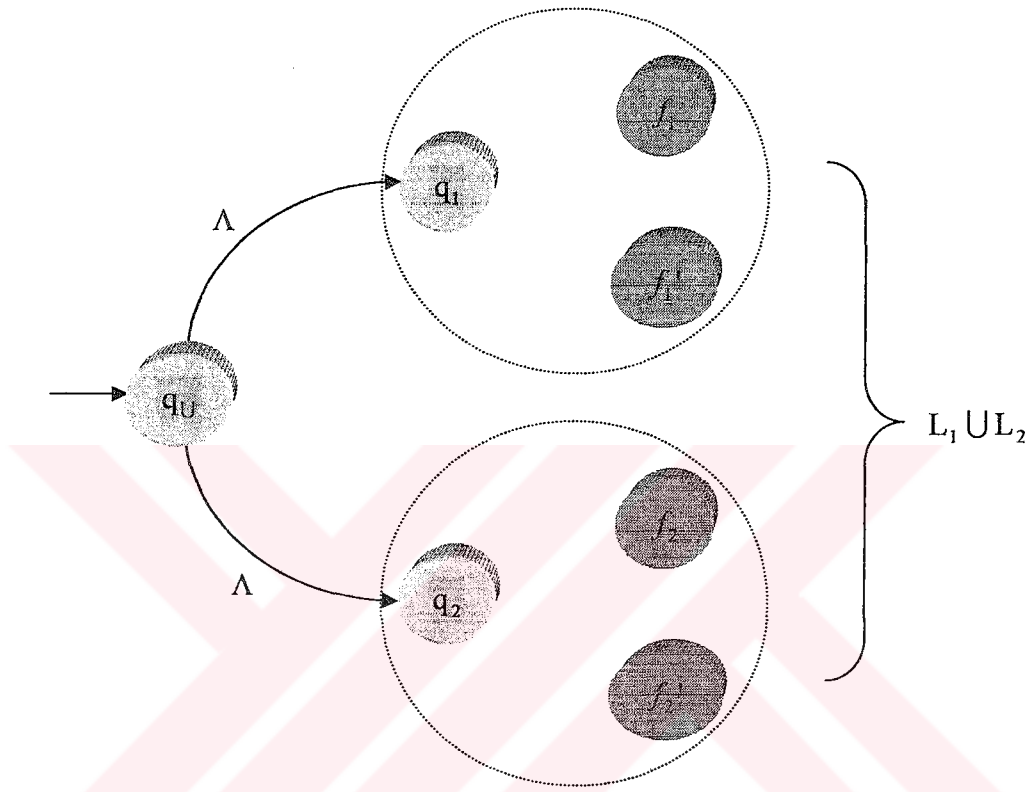
$$=\Lambda(\delta(B,1)\cup\delta(D,1)\cup\delta(E,1))$$

$$=\Lambda(\{E\}\cup\{D\}\cup\phi)$$

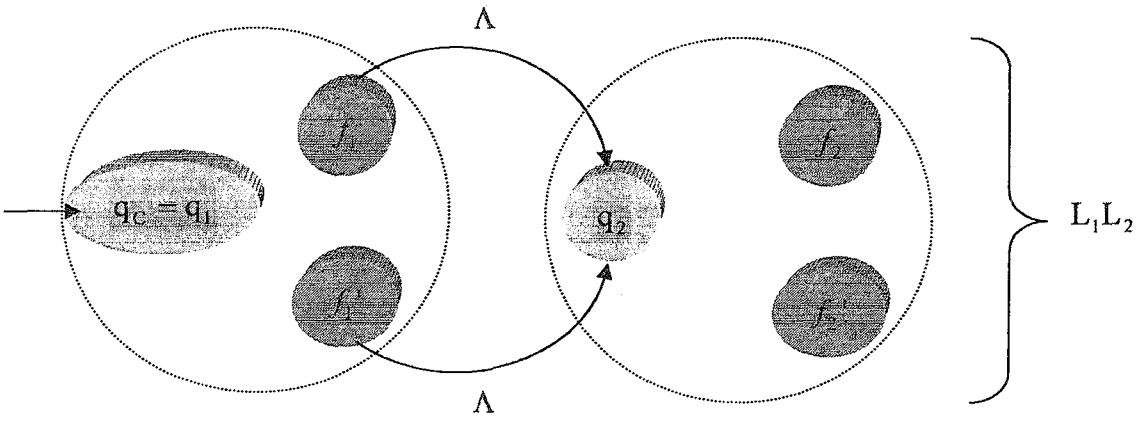
$$=\Lambda(\{E,D\})$$

= {E,D} E Kabul durumu kümede olduğundan x katarı KABUL edildi.

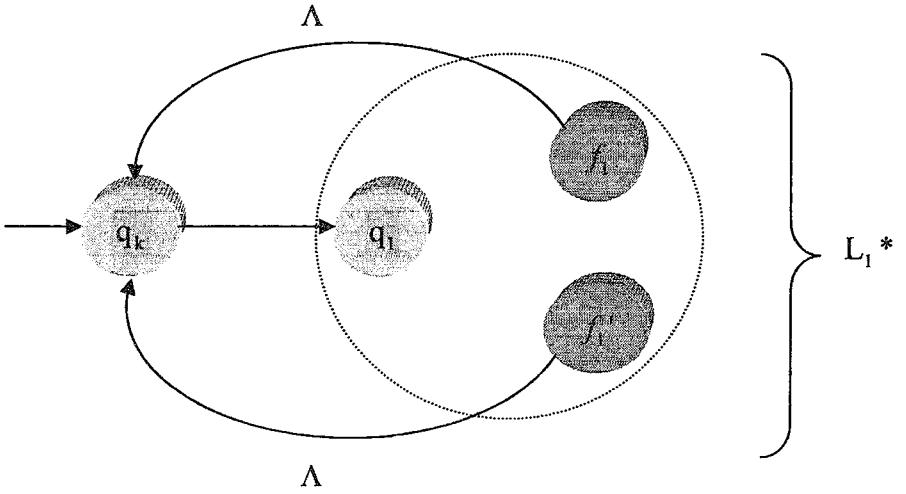
4.3.8. NFA - Λ Makinesinde Birleşim, Ekleme Ve Kleene* İşlemleri



Şekil 4.14. Verilen iki Makinenin $L_1 \cup L_2$ işlemi



Şekil 4.15. Verilen iki Makinenin $L_1 L_2$ işlemi



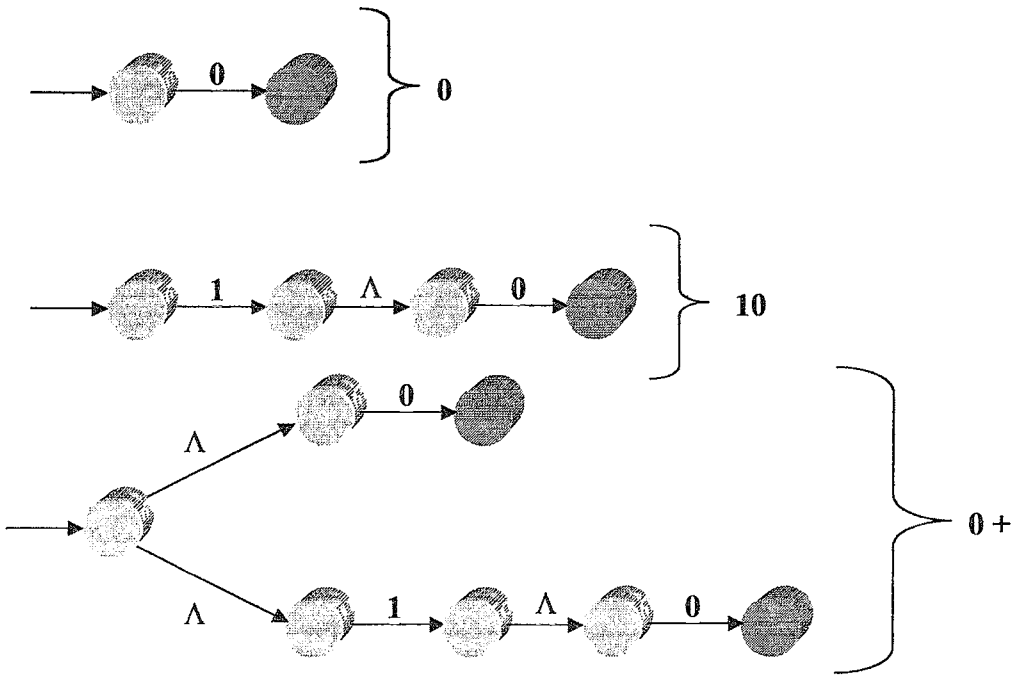
Şekil 4.16. Verilen iki Makinenin L_1 * işlemi

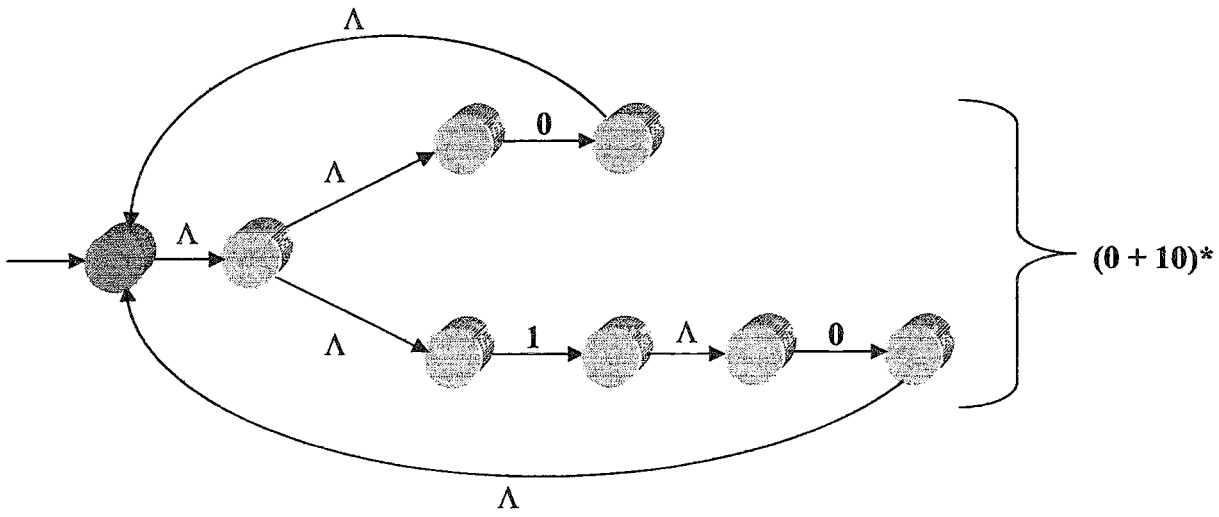
4.3.9. Verilen Bir Regüler İfadenin NFA - Λ Diyagramını Elde Etme

Örnek : (0 + 10)* (01)* regüler ifadesinin NFA Λ diyagramını elde etme.

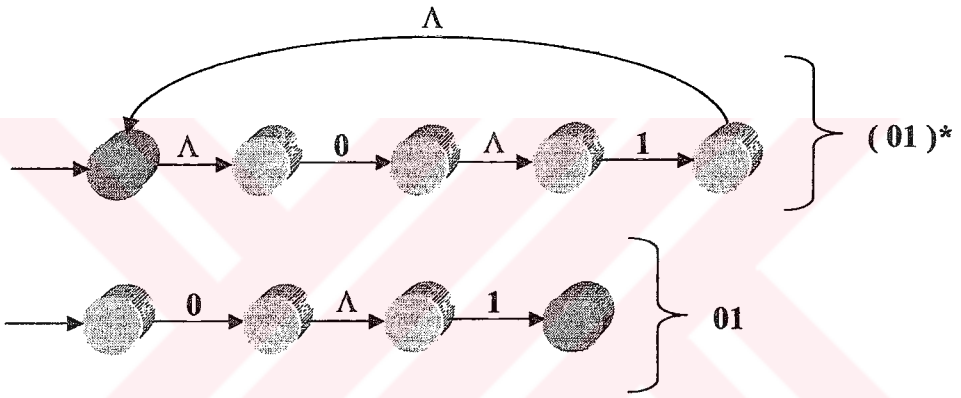
Temel 0 ve 1 regüler ifadesinin NFA Λ diyagramını elde edelim.

(0+10)* regüler ifadesini elde edelim.

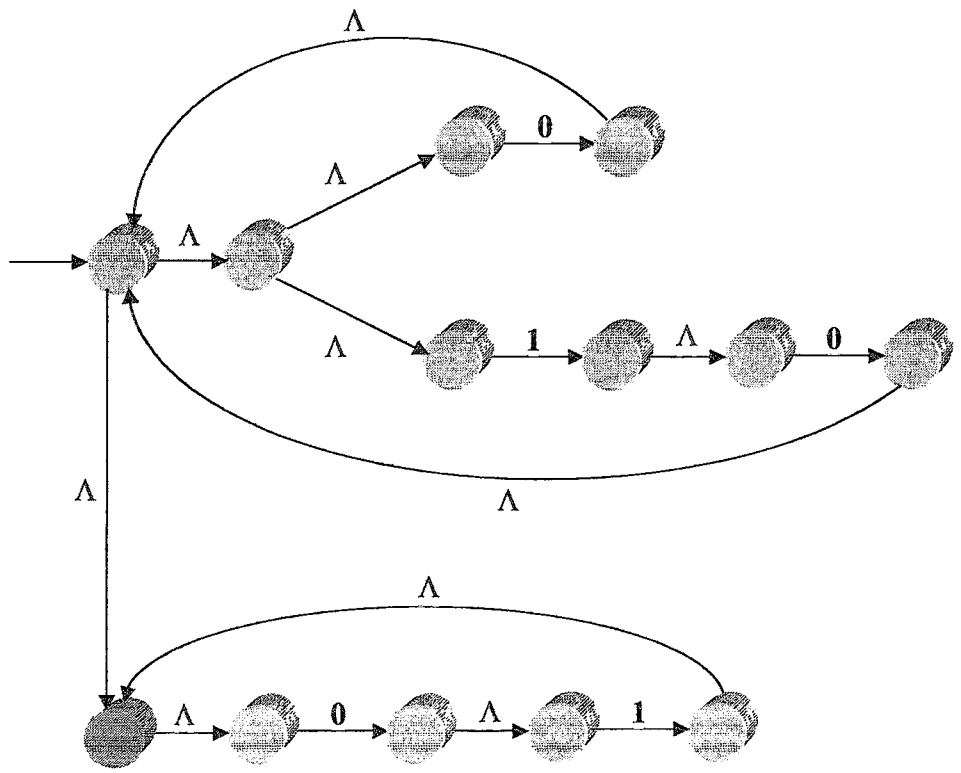




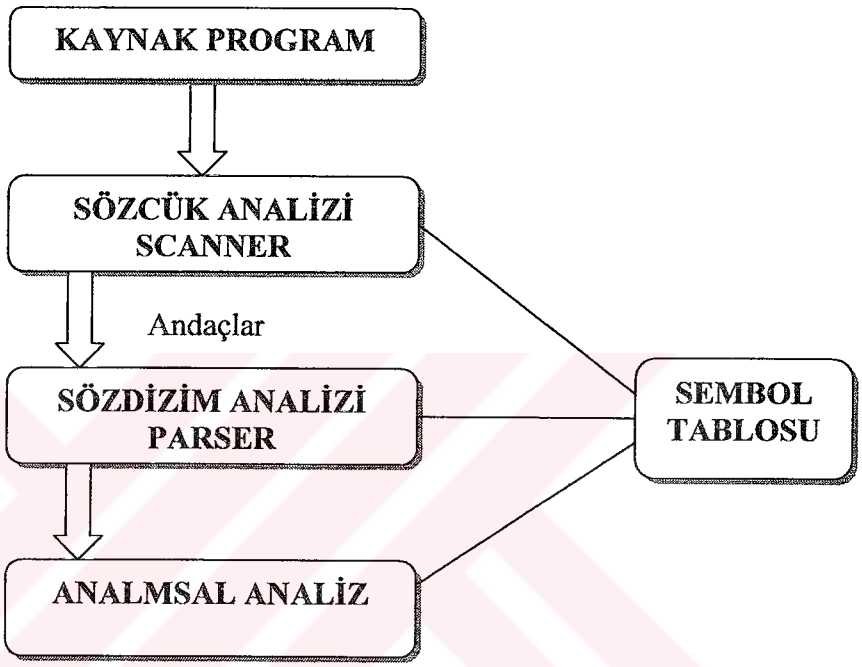
(01)* elde edelim



(0 + 10)* (01)* regüler ifadesini elde edelim.



BÖLÜM 5. SÖZDİZİM ANALİZİ



Şekil 5.1. Sözdizim Analizi

Sözdizim Analizi verilen bir kaynak program üzerinde sözdizimsel analizi yapar. Yani dilin kuralları içerisinde yer alan ve Sözcük Analizi tarafından da kabul edilmiş andaçların birleşip anlamlı cümleler kurması için gereken sözdizim yapılarının tanıtımının yapıldığı ve kontrol edildiği yerdir. Bu amaçla Sözdizim Analiz'e Sözcük Analizi tarafından kaynak programa ait andaçlar gelir. Sözcük Analizi kaynak programa ait tüm kelimeleri tanımlanan dilin kuralları içerisinde olup olmadığını kontrol eder ve Andaç Tablosu oluşturur.

Sözdizim Analiz bu Andaç Tablosunu kullanarak kaynak programa ait sözdizim yapısını verilen dilin kuralları çerçevesinde olup olmadığını analiz eder. Sözdizim Analiz dilin sözdizim yapısını tanımlamak için CFG'yi ve dilin kuralları

çerçevesinde geçerli olup olmadığını belirlemek için de otomat olarak PDA'yı kullanır.

5.1. Context Free Grammer - CFG

TANIM : $G = (V, \Sigma, S, P)$ beşlisinden oluşan gramerdir.

V : Değişkenler kümesi
Σ : Terminaller
S : Başlangıç sembolü veya değişkeni
P : Üretimler

CFG bir programlama dilinin programlarının sözdizimsel yapılarını tanımlamak için kullanılır.

Aşağıda basit bir CFG yer almakta.

$S \rightarrow AB \mid ASB$
$A \rightarrow a$
$B \rightarrow b$

Verilen CFG'ye ait bilgiler aşağıdaki gibi olur;

$V = \{S, A, B\}$: Değişkenler kümesi
$\Sigma = \{a, b\}$: Terminaller
$S = \{S\}$: Başlangıç sembolü veya değişkeni
$P = \{S \rightarrow AB \mid ASB, A \rightarrow a, B \rightarrow b\}$: Üretimler

Burada S Non-Terminaline ait iki farklı üretim biçimi olduğundan | sembolü ile bu üretimler ayrılmıştır. Yani | sembolü bir Non-Terminale ait birden fazla olan üretimleri göstermek için kullanılır.

5.1.1. Türetme

Bir CFG de üretim verilen üretim kuralları uygulanarak yapılır. Amaç verilen giriş bilgisini bu üretim kurallarını düzgün sırada kullanarak elde etmektir. Çünkü türetme esnasında üretim kurallarının hangisinin uygulanması gerektiğine karar vermek gerekir. Aksi takdirde istenen giriş bilgisi CFG tarafından üretilemeyebilir. Verilen CFG'de türetme başlangıç sembolüne veya değişkenine ait olan üretim kurallarından başlanarak yapılır.

Örnek olarak yukarıdaki gramer içerisinde tanımlı `aabb' girişini inceleyelim:

Uygulanan Kural	Üretilen
S	
$S \rightarrow ASB$	ASB
$A \rightarrow a$	aSB
$B \rightarrow b$	aSb
$S \rightarrow AB$	$aABb$
$A \rightarrow a$	$aaBb$
$B \rightarrow b$	$aabb$

Üretim bazen birden çok şekilde yapılabilir. Aynı giriş bilgisini farklı üretim kuralları uygulayarak çözmeye çalışalım.

Uygulanan Kural	Üretilen
S	
$S \rightarrow ASB$	ASB
$B \rightarrow b$	ASb
$A \rightarrow a$	aSb
$S \rightarrow AB$	$aABb$
$B \rightarrow b$	$aAbb$
$A \rightarrow a$	$aabb$

5.1.2. LeftMost ve RightMost Türetme

LeftMost Türetme her zaman en soldaki Non-Terminal seçilir.
RightMost Türetme her zaman en sağdaki Non-Terminal seçilir.

Örnek:

$$\exp r \rightarrow \exp r OP \exp r$$
$$\exp r \rightarrow (\exp r)$$
$$\exp r \rightarrow -\exp r$$
$$\exp r \rightarrow id$$
$$OP \rightarrow + \mid - \mid * \mid /$$

LeftMost Türetim

$$\exp r$$
$$-\exp r$$
$$-(\exp r)$$
$$-(\exp r OP \exp r)$$
$$-(id OP \exp r)$$
$$-(id + \exp r)$$
$$-(id + id)$$

RightMost Türetim

$$\exp r$$
$$-\exp r$$
$$-(\exp r)$$
$$-(\exp r OP \exp r)$$
$$-(\exp r OP id)$$
$$-(\exp r + id)$$
$$-(id + id)$$

5.1.3. Üretim Ağacı

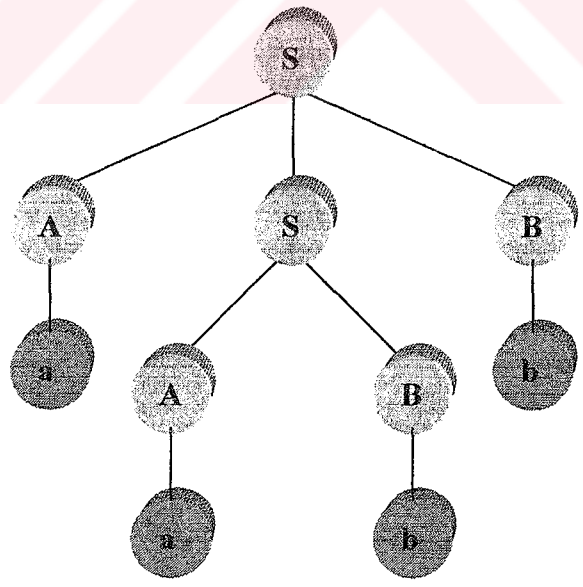
Verilen bir CFG de üretim ağacının önemi çok büyüktür. Karmaşık bir CFG de verilen bir girişin kurallara uygun üretimi için bu ağaçtan yararlanılır. Çünkü karmaşık bir CFG de yanlış bir üretim yapmak giriş bilgisinin üretimini ortadan kaldırabilir.

5.1.4. Parse Ağacı

CFG'de verilen üretim kurallarına göre istenen girişi sağlayan üretim ağacına Parse Ağacı denir.

$S \rightarrow AB \mid ASB$
$A \rightarrow a$
$B \rightarrow b$

Yukarıdaki örnek CFG kurallarına göre üretilen 'aabb' girişine ait parse ağacı aşağıda verilmiştir.



Şekil 5.2. Parse Ağacı

Parse Ağacının Yapısı:

- Kök CFG nin başlangıç sembolü olmak zorundadır.
- En alt dallarda her zaman terminaller yer alır.
- İç düğümler gramere ait non-terminallerden oluşur.
- Eğer giriş dil tarafından kabul edilmiş ise parse ağacı her zaman oluşturulabilir.

5.1.5. Belirsizlik

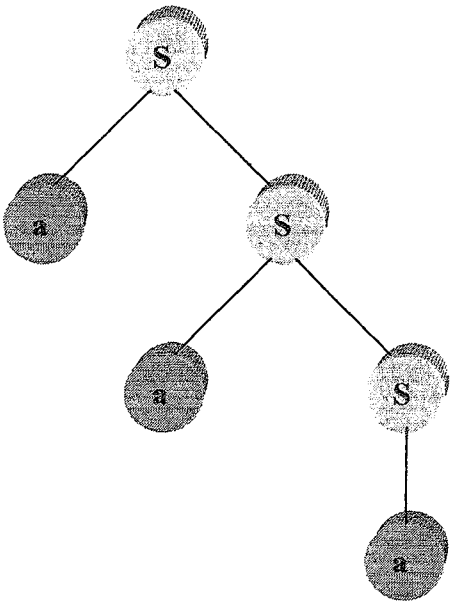
Bir CFG de birden fazla üretim varsa buna bu gramer de Belirsizlik var denir. Bu üretim biçimlerinin kullanımına göre üretimler ve parse ağacı farklı yaratılacaktır. CFG'de belirsizlik verilen giriş bilgisinin üretim kuralları uygulanarak gerçeklenmesini zorlaştırır çünkü hangi üretim kuralının uygulanması gerektiği kararını vermek gerekir.

Aşağıda belirsizliğe neden olan bir CFG verilmiştir.

$S \rightarrow aS \mid Sa \mid a$

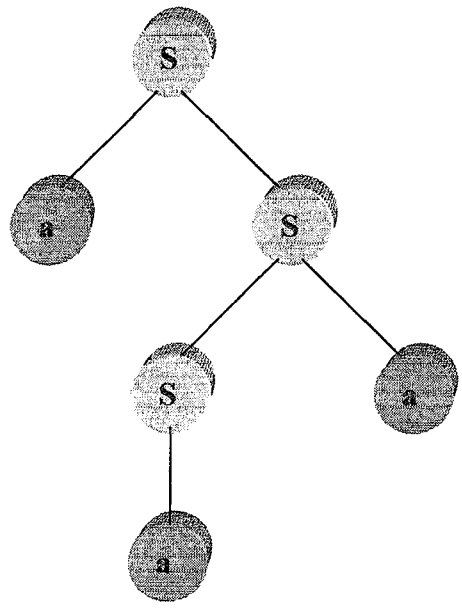
Giriş bilgisi 'aaa' olsun. Buna göre 4 farklı üretim ağacı elde edilir.

Uygulanan Kural	*Üretilen*
	S
$S \rightarrow aS$	aS
$S \rightarrow aS$	aaS
$S \rightarrow a$	aaa



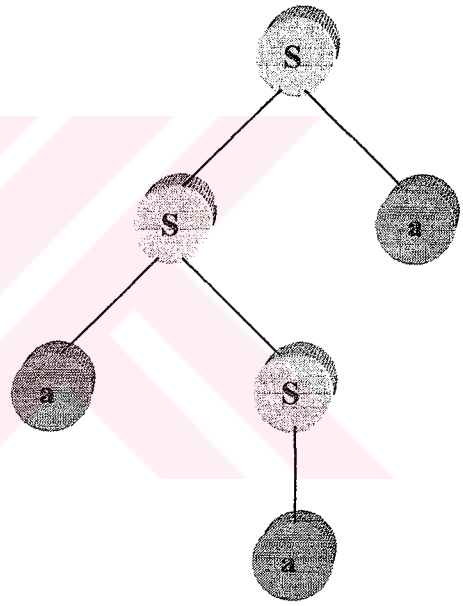
Şekil 5.3. Parse Ağacı-1

Uygulanan Kural	Üretilen
	S
$S \rightarrow aS$	aS
$S \rightarrow Sa$	aSa
$S \rightarrow a$	aaa



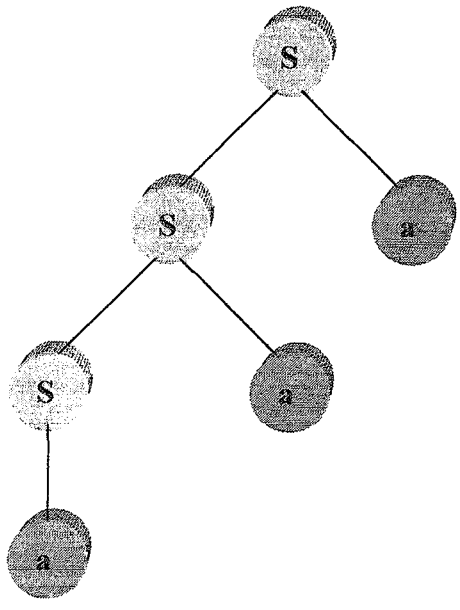
Şekil 5.4. Parse Ağacı-2

Uygulanan Kural	*Üretilen*
	S
$S \rightarrow Sa$	Sa
$S \rightarrow aS$	aSa
$S \rightarrow a$	aaa



Şekil 5.5. Parse Ağacı-3

Uygulanan Kural	*Üretilen*
	S
$S \rightarrow Sa$	Sa
$S \rightarrow Sa$	Saa
$S \rightarrow a$	aaa



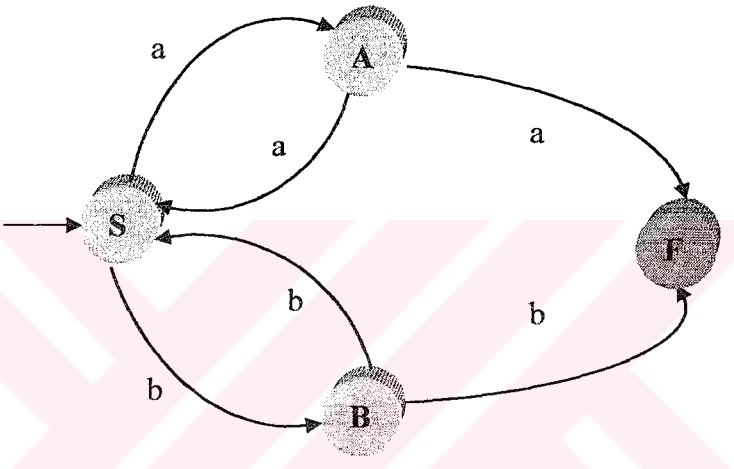
Şekil 5.6. Parse Ağacı-4

5.1.6. CFG'den NFA elde etme

Örnek CFG;

$S \rightarrow aA \mid bB$
$A \rightarrow aS \mid a$
$B \rightarrow bS \mid b$

CFG'den NFA elde edilmesi



Şekil 5.7. NFA diyagramı

5.2. Push Down Automata - PDA

TANIM : $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ yedilisinden oluşur. Bunlar ;

Q : Durumlar kümesi

Σ : Giriş Alfabesi

Γ : Yığın Alfabesi

q_0 : Başlangıç Durumu

Z_0 : Başlangıç Sembolü

A : Kabul Durumları

δ : Geçiş Fonksiyonu $Qx(\Sigma \cup \{\Lambda\} x \Gamma \rightarrow (Qx\Gamma^*)$

5.2.1. PDA Geçiş Fonksiyonu

FA Geçiş Fonksiyonu : $\delta : Q x \Sigma \rightarrow Q$ $\delta(q,a) = p$

PDA Geçiş Fonksiyonu : $\delta : Q x \Sigma x \Gamma \rightarrow Q x \Gamma^*$ $\delta(q,a,x) = (p,\alpha)$

q : Mevcut Durum

p : Geçiş Durumu

a : Giriş Sembolü

x : Yığın Sembolü

TANIM : $x \in \Sigma^*$ ve $(q_0, x, Z_0) \overset{*}{\underset{M}{\vdash}} (q, A, \alpha)$ $\alpha \in \Gamma^*$ ve $a \in A$ ise x katarı kabul edilmiştir. L=L(M)

5.2.2. Deterministic Push Down Automata - DPDA

TANIM : $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ yedilisinden oluşur. Bunlar ;

Q : Durumlar kümesi

Σ : Giriş Alfabesi

Γ : Yığın Alfabesi

q_0 : Başlangıç Durumu

Z_0 : Başlangıç Sembolü

A : Kabul Durumları

δ : Geçiş Fonksiyonu $Qx(\Sigma \cup \{\Lambda\} x \Gamma \rightarrow (Q x \Gamma^*)$

Bir PDA'nın DPDA olabilmesi için aşağıdaki şartları sağlaması gerekir.

- Herhangi $q \in Q, a \in \Sigma \cup \{\Lambda\}$ ve $x \in \Gamma$ için $\delta(q,a,x)$ kümesi en fazla 1 elemanlı olmalıdır.

- Herhangi $q \in Q$, $x \in \Gamma$ için $\delta(a,\Lambda,x) \neq \phi$ ise $\delta(q,a,x) = \phi$ olmalıdır.

5.2.3. CFG'den PDA elde etme

TEOREM : $G=(V,\Sigma,S,P)$ bir CFG olsun. Buna göre L(M)=L(G) olan bir PDA vardır.

CFG için $G=(V,\Sigma,S,P)$ beşlisinden oluşan gramerdir.

V : Değişkenler kümesi
Σ : Terminaller
S : Başlangıç sembolü veya değişkeni
P : Üretimler

PDA için $M=(Q,\Sigma,\Gamma,q_0,Z_0,A,\delta)$

$Q=\{q_0,q_1,q_2\}$: Durumlar kümesi
$\Gamma=V\cup\Sigma\cup\{Z_0\}$: Yığın Alfabesi
$A=\{q_2\}$: Kabul Durumu

1. $\delta(q_0,\Lambda,Z_0)=\{(q_1,SZ_0)\}$
2. Her $A\in V$ için $\delta(q_1,\Lambda,A)=\{(q_1,\alpha)\,|\,A\to\alpha$ tüm üretimler$\}$
3. Her $a\in\Sigma$ için $\delta(q_1,a,a)=\{(q_1,\Lambda)\}$
4. $\delta(q_1,\Lambda,Z_0)=\{(q_2,Z_0)\}$

ÖRNEK : $L=\{x\in\{a,b\}^*\,|\,n_a(x)>n_b(x)\}$

$S\to a\,|\,aS\,|\,bSS\,|\,SSb\,|\,SbS$ verilsin.

Verilen CFG'den PDA'yı elde edelim. Bunun için yukarıdaki adımları teker teker uygulamamız gerekiyor.

$M=(Q,\Sigma,\Gamma,q_0,Z_0,A,\delta)$

$M=(\{q_0,q_1,q_2\},\{a,b\},\{S,a,b,Z_0\},q_0,Z_0,\{q_2\},\delta)$

Tablo 6.1. PDA Tablosu

Durum	Giriş	Yığın	Hareket	Adım
q_0	Λ	Z_0	(q_1, SZ_0)	1
q_1	Λ	S	$(q_1, a), (q_1, aS), (q_1, bSS), (q_1, SSb), (q_1, SbS)$	2
q_1	a	a	(q_1, Λ)	3
q_1	b	b	(q_1, Λ)	3
q_1	Λ	Z_0	(q_1, SZ_0)	4

5.3. Parsing

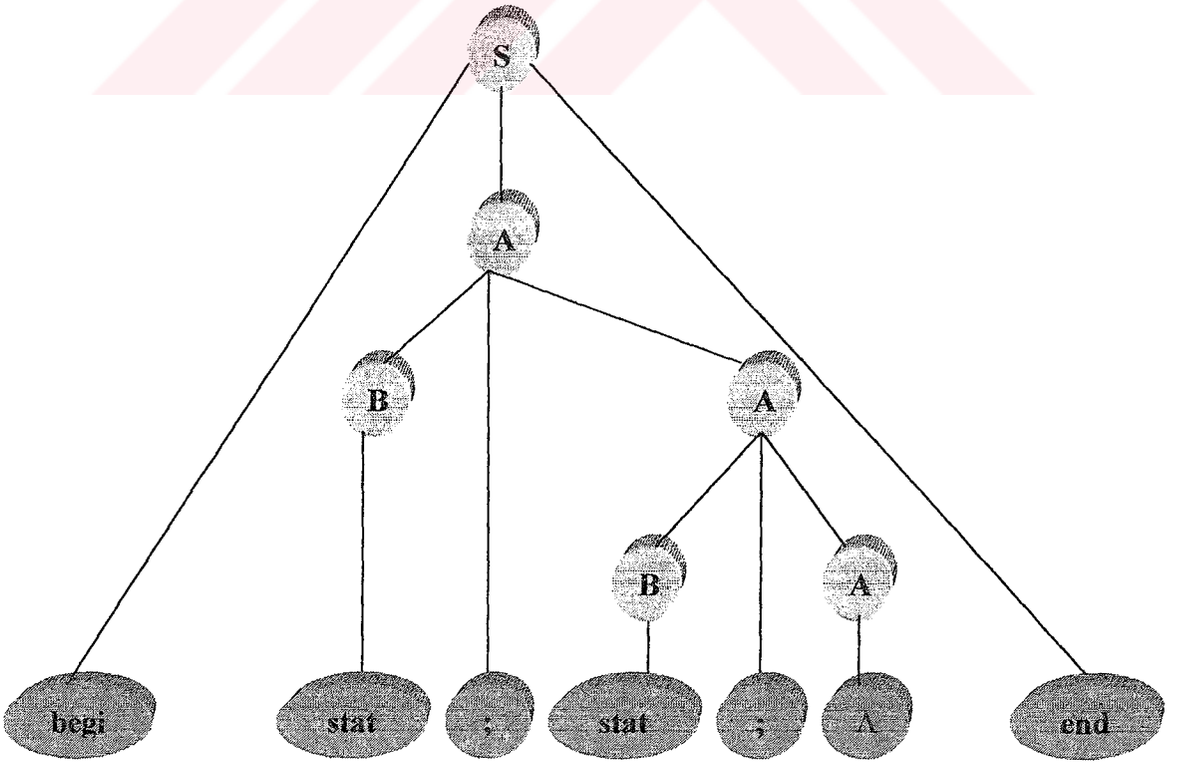
Aşağıdaki CFG verilmiş olsun;

$S \rightarrow beginAend\$$

$A \rightarrow B;A \mid \Lambda$

$B \rightarrow stat \mid beginAend$

giriş bilgisi olarak verilen "*begin stat ; stat ; end*" katarını verilen CFG'den üretilen parse ağacı aşağıdaki gibi olur;



5.3.1. Top-Down LL(k) Parsing

Top-Down LL(k) Parsing'de üretime verilen CFG'de başlangıç sembolünden yani kök düğümden başlanarak istenen terminaller üretilir. Yani üretime kökten başlanır en alttaki dallara yani terminallere ulaşılarak sonlanır.

İlk L sembolü giriş bilgisini soldan sağa doğru tarandığını gösterir.
İkinci L sembolü üretimin LeftMost olduğunu,
k ise her üretim adımında çeşitli türetimleri seçmek için kullanılır.

Örnek olarak LL(1) de ileriye dönük olarak 1 adım incelenmektedir.

Genel olarak bir Top-Down LL(k) Parsing başlangıç sembolünden başlayarak bu sembole ait üretimlerden kaynak programda verilen giriş bilgisine ait terminalleri elde etmeye çalışan bir yaklaşıma sahiptir.

Aşağıda yukarıdaki örneğe ait bir Top-Down LL(k) Parsing üretimi yer almakta;

Tablo 6.2. Top-Down LL(k) Parsing

Uygulanan Kural	*Üretilen*
	S
$S \rightarrow beginAend\$$	$beginAend\$$
$A \rightarrow B;A$	$beginB;Aend\$$
$B \rightarrow stat$	$beginstat;Aend\$$
$A \rightarrow B;A$	$beginstat;B;Aend\$$
$B \rightarrow stat$	$beginstat;stat;Aend\$$
$A \rightarrow \Lambda$	$beginstat;stat;end\$$

5.3.2. Bottom-Up LR(k) Parsing

Bottom-Up (LR) Parsing'de üretime CFG,de verilen terminalleri üretmek ile başlanır ve kök düğüme ulaşınca sonlanır. Yani Top-Down (LL) Parsing'in tam tersi bir akış izlenir. Önce terminaller ve bunları kullanan Non-Terminaller üretilir ardından kök düğüme yani başlangıç sembolüne ulaşılır.

L sembolü giriş bilgisini soldan sağa doğru tarandığını gösterir.
R sembolü üretimin RightMost olduğunu,
k ise her üretim adımında çeşitli türetimleri seçmek için kullanılır.

Örnek olarak LR(1) de ileriye dönük olarak 1 adım incelenmektedir.

Genel olarak bir Bottom-Up LR(k) Parsing verilen giriş bilgisinde yer alan terminalleri elde etmek için CFG'de tanımlanan üretim kurallarını kullanma ve kök yani aşlangıç sembolüne elde etmeye çalışan bir yaklaşıma sahiptir.

Aşağıda yukarıdaki örneğe ait bir Bottom-Up (LR) Parsing üretimi yer almakta;

Tablo 6.3. Bottom-Up LR(k) Parsing

Uygulanan Kural	*Üretilen*
	beginstat; *stat*; *end*$
$B \rightarrow stat$	*beginB*; *stat*; *end*$
$B \rightarrow stat$	*beginB*; *B*; *end*$
$A \rightarrow \Lambda$	*beginB*; *B*; *Aend*$
$A \rightarrow B; A$	*beginB*; *Aend*$
$A \rightarrow B; A$	*beginAend*$
	S

Yukarıda verilen tabloda verilen giriş bilgisinde yer alan terminalleri CFG'de tanımlanan üretim kuralları kullanarak elde edilmiştir. Yani verilen bu giriş bilgisi sözdizim tanımı olarak verilen CFG tarafından dolayısı ile tanımlanan dil tarafından kabul edilmiştir.

5.3.3. First Kümesi

First(A), A Non-Terminaline ait üretimler içerisindeki ilk terminallerin oluşturduğu kümedir. Yani FIRST kümesi terminallerden oluşan bir kümedir ve ilgili Non-Terminale ait ilk başlangıç terminallerini verir.[7]

Örnek olarak aşağıdaki gibi bir CFG verilsin;

$S \rightarrow Ax \mid By \mid c$
$A \rightarrow a$
$B \rightarrow b$

$FIRST(S) = FIRST(A) \cup FIRST(B) \cup "c"$
$FIRST(S) = \{a, b, c\}$
$FIRST(A) = \{a\}$
$FIRST(B) = \{b\}$

elde edilir.

FIRST kümesini hesaplama Algoritması

1. FIRST(A) içeriğini temizle
2. A Non-Terminaline ait üretimlerin ilk terimini al
3. Eğer ilk terim terminal ise FIRST(A)'ya ekle
4. Eğer Non-Terminal ise ilgili Non-Terminale ait FIRST kümesini FIRST(A)'a ekle.
5. Eğer A üretimi varsa FIRST(A)'ya ekle.

Örnek olarak aşağıda verilen CFG üzerinde bu algoritmayı uygulayalım;

$S \rightarrow aSe \mid B$

$B \rightarrow bBe \mid C$

$C \rightarrow cCe \mid d$

elde edilen FIRST kümeleri aşağıdaki gibi olur.

$FIRST(C) = \{c, d\}$

$FIRST(B) = \{b\} \cup FIRST(C)$
$FIRST(B) = \{b\} \cup \{c, d\}$
$FIRST(B) = \{b, c, d\}$

$FIRST(S) = \{a\} \cup FIRST\{B\}$
$FIRST(S) = \{a\} \cup \{b\} \cup FIRST(C)$
$FIRST(S) = \{a, b\} \cup \{c, d\}$
$FIRST(S) = \{a, b, c, d\}$

5.3.4. Follow Kümesi

$FOLLOW(A) = \{a \in \Sigma \mid S \rightarrow +...Aa...\} \cup \{\$ \text{ eğer } S \rightarrow +...A \mid A \text{ başlangıç sembolü ise }\}$

şeklinde ifade edilir. [7]

FOLLOW(A) Kümesini hesaplama

1. Eğer A başlangıç sembolü ise FOLLOW(A)'ya $ koy.
2. Eğer A üretimin son terimi ise FOLLOW(A)'ya $ koy.
3. Eğer a A Non-Terminalden sonra gelen terminal ise FOLLOW(A)'ya ekle.

Λ sembolü asla FOLOW kümesinin bir elemanı olamaz.

$S \rightarrow ABC \mid AD$

$A \rightarrow a \mid aA$

$B \rightarrow b \mid c \mid \Lambda$

$C \rightarrow DaC$

$D \rightarrow bb \mid cc$

$FIRST(D) = \{b, c\}$
$FIRST(C) = FIRST(D)$
$FIRST(C) = \{b, c\}$
$FIRST(B) = \{b, c, \Lambda\}$
$FIRST(A) = \{a\}$
$FIRST(S) = FIRST(A)$
$FIRST(A) = \{a\}$

$FOLLOW(S) = \{\$\}$
$FOLLOW(A) = \{b, c\}$
$FOLLOW(B) = \{b, c\}$
$FOLLOW(C) = \{\$\}$
$FOLLOW(D) = \{a, \$\}$

BÖLÜM 6. PROGRAMIN TANITILMASI

6.2. Programın Amacı

Program kullanıcı tarafından tanımlanan derleyiciye ait bir kaynak programın Sözcük Analizi ve Sözdizim Analiz işlemlerini gerçekleştirir.

Sözcük Analizi işlemi için Regüler diller otomat olarak ise Finite Automata kullanmaktadır. Sözdizim Analiz işlemi için CFG ve otomat olarak da Push-Down Automata kullanmaktadır.

6.1.1. Sözcük Analizi

Sözcük Analizi için verilen bir programlama diline ait kaynak kodunu bu programlama diline ait,

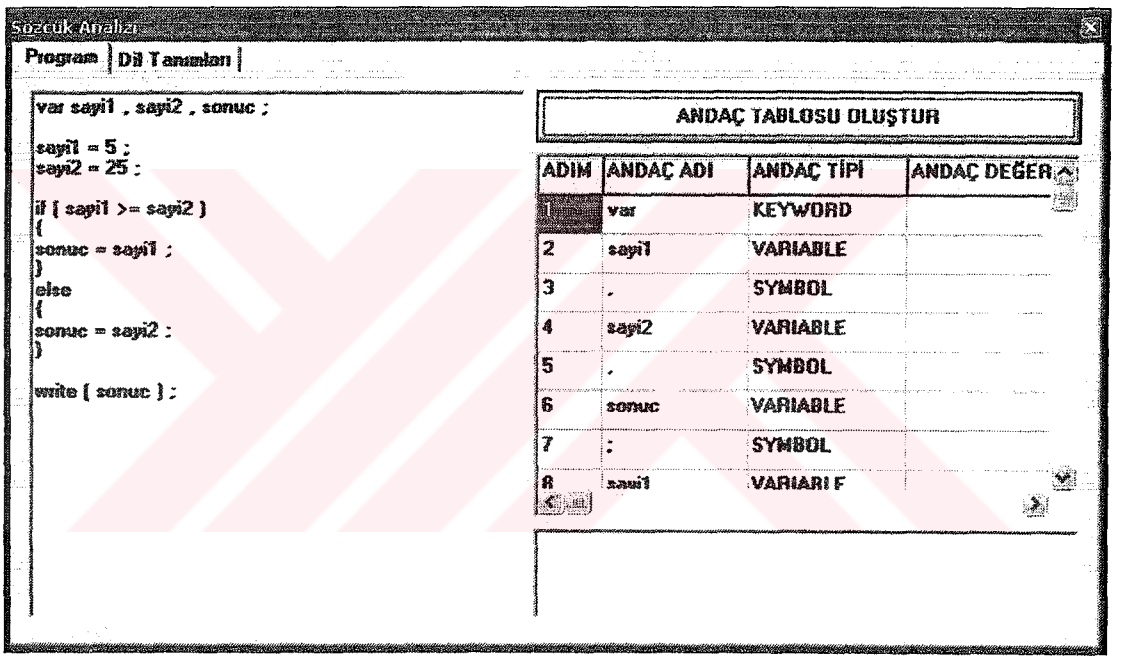
- Keywordler
- Operatörler
- Özel Semboller
- Sabitler
- Tipler
- Fonksiyonlar
- Tanımlayıcılar

kullanarak andaçlarına parçalar.
Bu işlem için yukarıda tanımlanan her bir yapının NFA diyagramını oluşturur ve verilen kaynak programa ait her bir kelime için bu yapılardan hangisine ait olduğu

tespit edilir. Bu işlem sonunda Andaç Tablosu elde edilir. Andaç Tablosunda aşağıdaki bilgiler yer almaktadır;

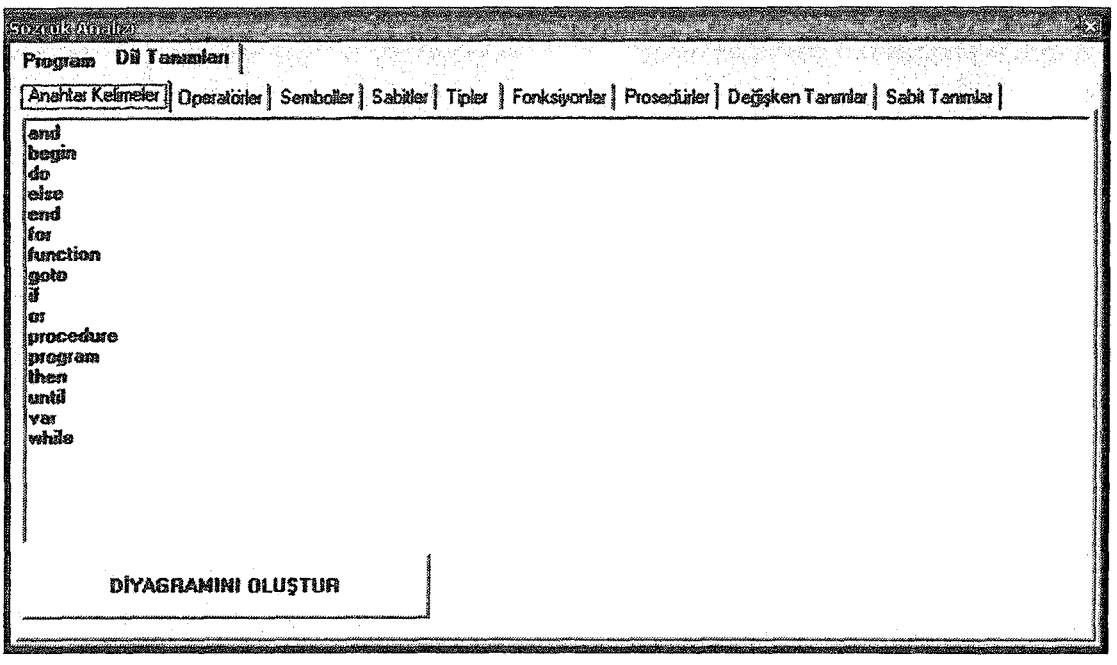
- Andaç adı
- Andaç tipi
- Andaç değeri

Bu şekilde verilen kaynak program içerisinde ait olduğu programlama dilinin kurallarına göre kelime yapıları incelenir.



Şekil 6.1 Sözcük Analizi Penceresi

Yukarıda Sözcük Analizi Penceresi verilmektedir. Sözcük Analizi Penceresi kullanıcının kaynak programı girişini ve bu kaynak program üzerinde Sözcük Analizi yapmasını sağlar.



Şekil 6.2. Sözcük Analizi - Dil Tanımları Penceresi

Yukarıda Sözcük Analizi için gerekli Dil Tanımlamaları yer almaktadır. Daha önce de bahsedildiği gibi bir dilin sözcüksel parçaları

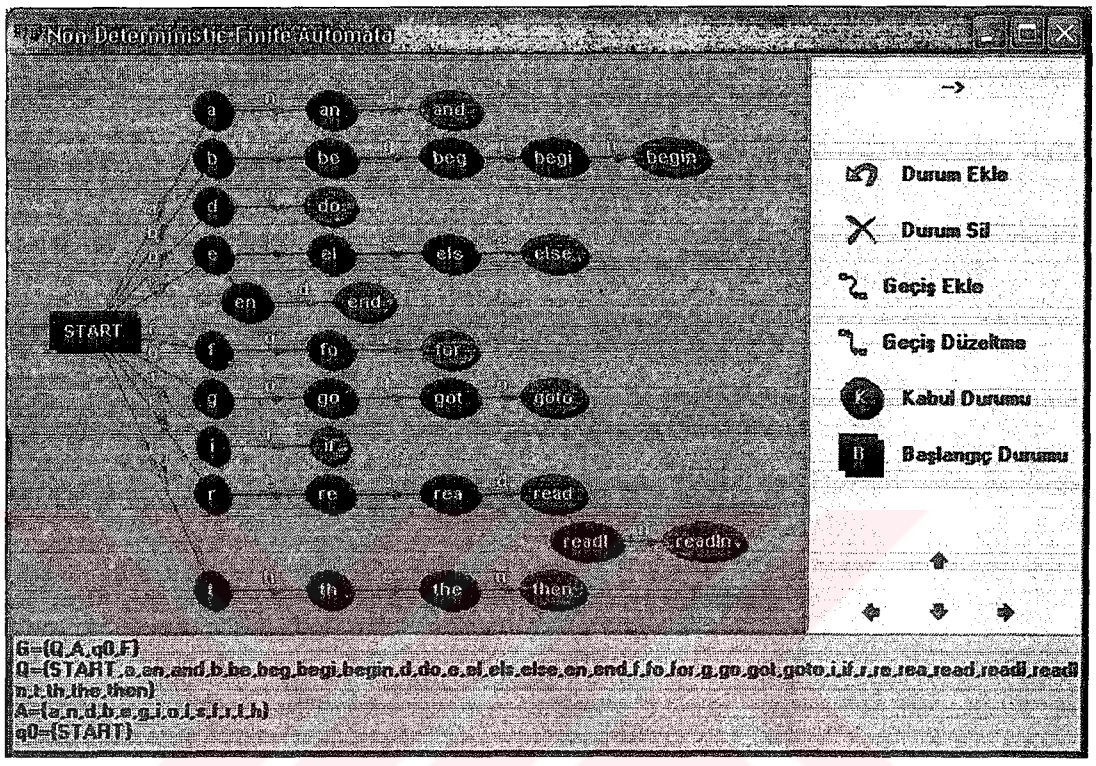
- Programlama diline ait anahtar kelimeler
- Operatörler
- Özel semboller
- Sabitler
- Tipler
- Fonksiyonlar ve tanımlayıcılar.

olarak verilebilir.

Yukarıda ki görüntüden de anlaşılacağı gibi "Dil Tanımları Penceresinde" kullanıcı tarafından bir dilin tanımlanmasına olanak tanınmaktadır.

Kullanıcı tarafından girilen tanımlar ile oluşturulan dil verilen kaynak program üzerinde kullanılmaktadır.

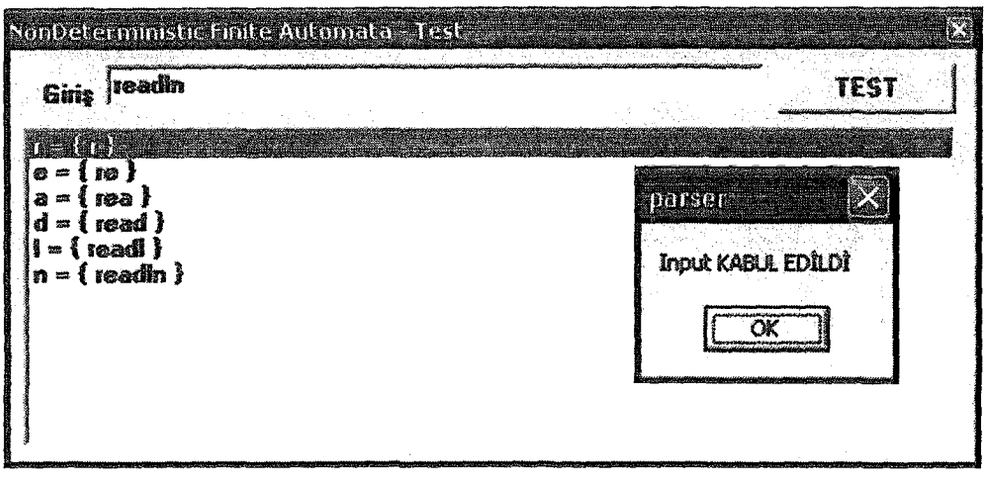
Yukarıdaki pencereye ait NFA diyagramı aşağıdaki gibi olur. Burada kırmızı renkteki durumlar "Kabul Durumlarını" dikdörtgen şeklindeki durum başlangıç durumunu diğer durumlar da normal durumları göstermektedir.



Şekil 6.3. Non-Deterministic Finite Automata - Keywordlerin tanımlanması

Burada ilgili tüm kelimeler için NFA durumları yaratılmıştır. İstenen herhangi bir giriş bilgisi bu makine üzerinde test edilerek dilin yapısına uyup uymadığı belirlenebilir.

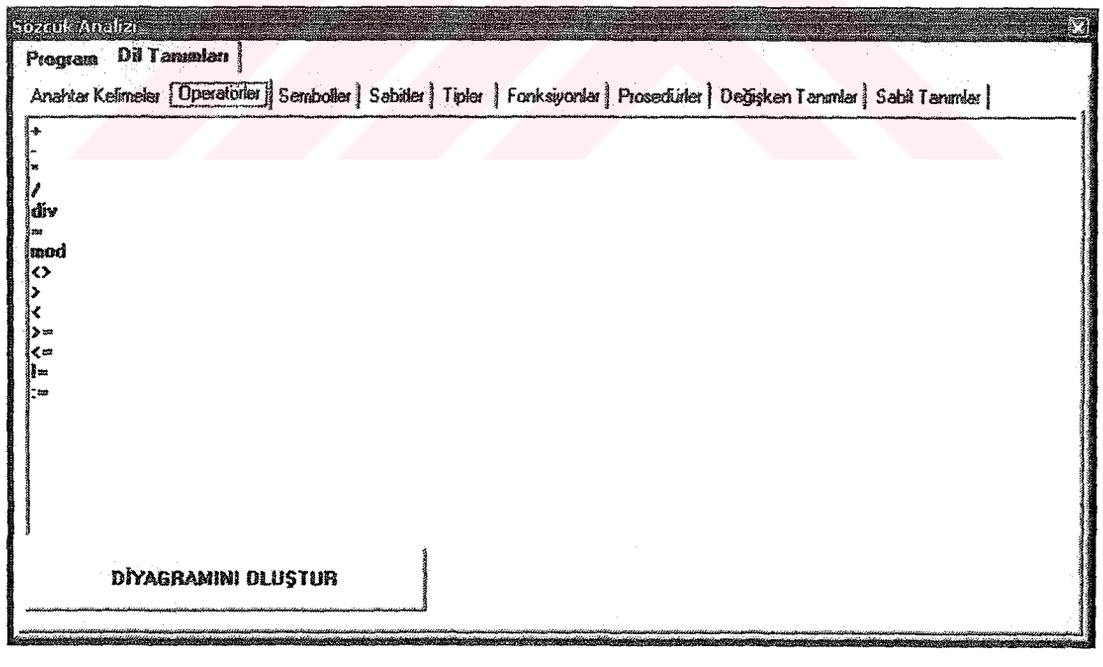
Aşağıda kullanıcı tarafından girilen dile ait "Keywordler"'in program tarafından oluşturulan diyagramına ait test penceresi yer almakta.



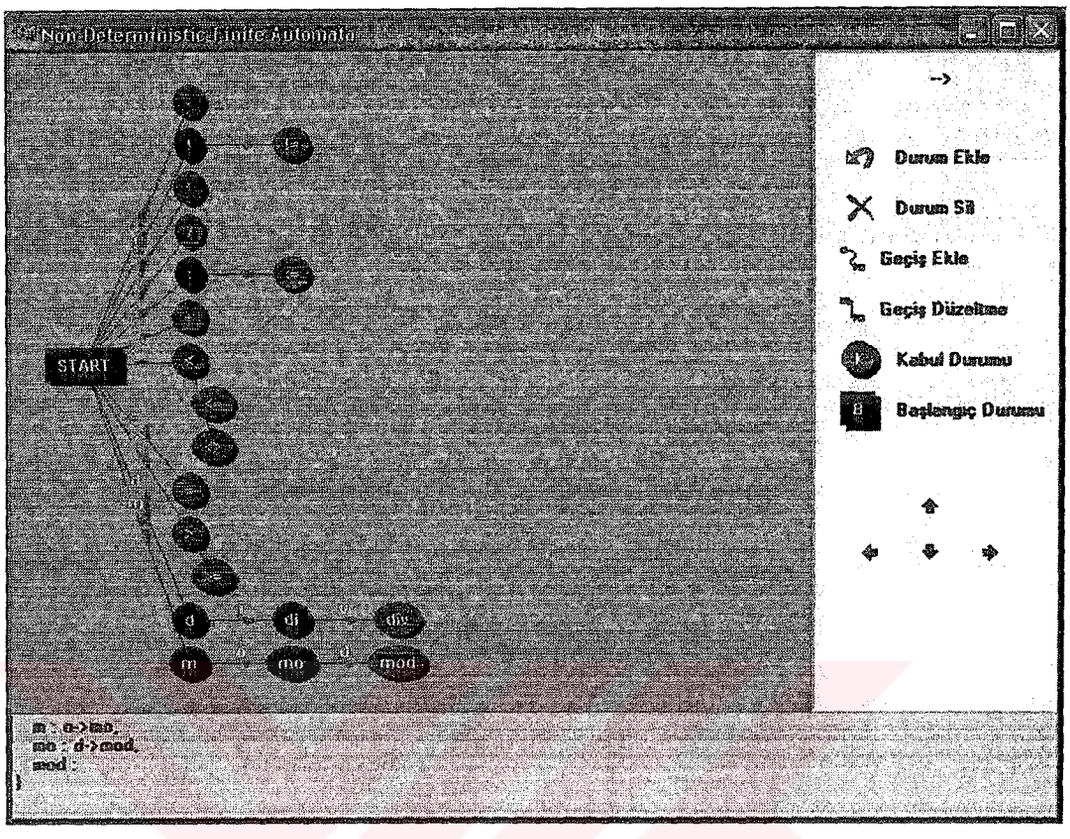
Şekil 6.4. Non-Deterministic Finiti Automata - Test Penceresinde Keyword testi

Örnek olarak "readln" girişini NFA makinesinde test işleminde program yukarıda ki gibi bir görüntü verecektir.

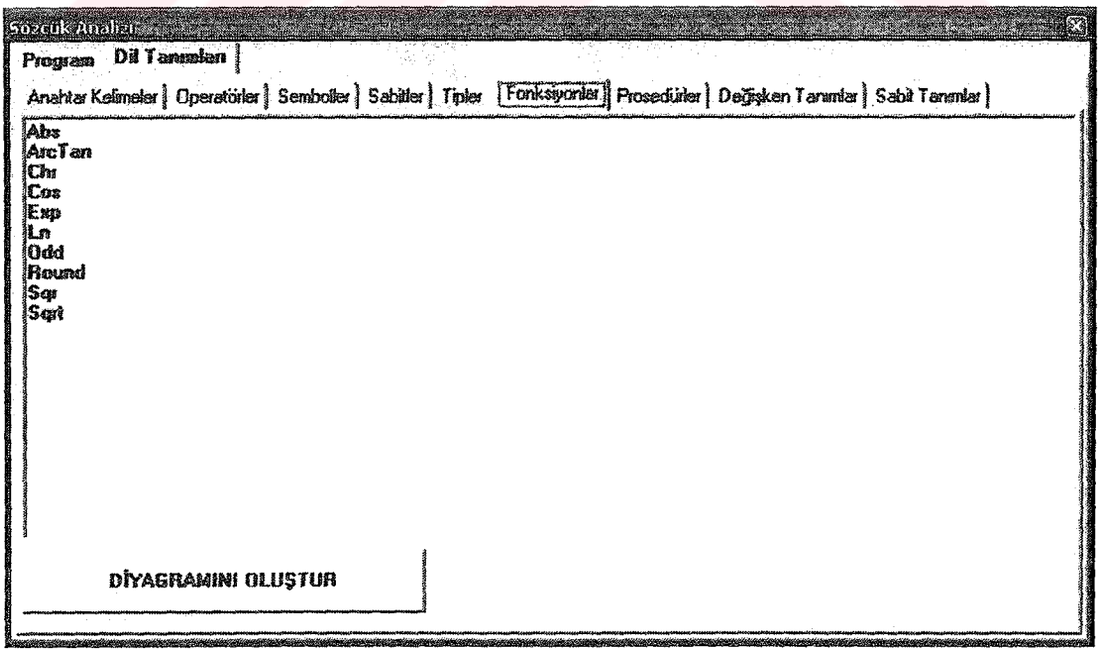
Aşağıda Sözcük Analizi – Operatör Penceresine ait görüntü yer almakta.



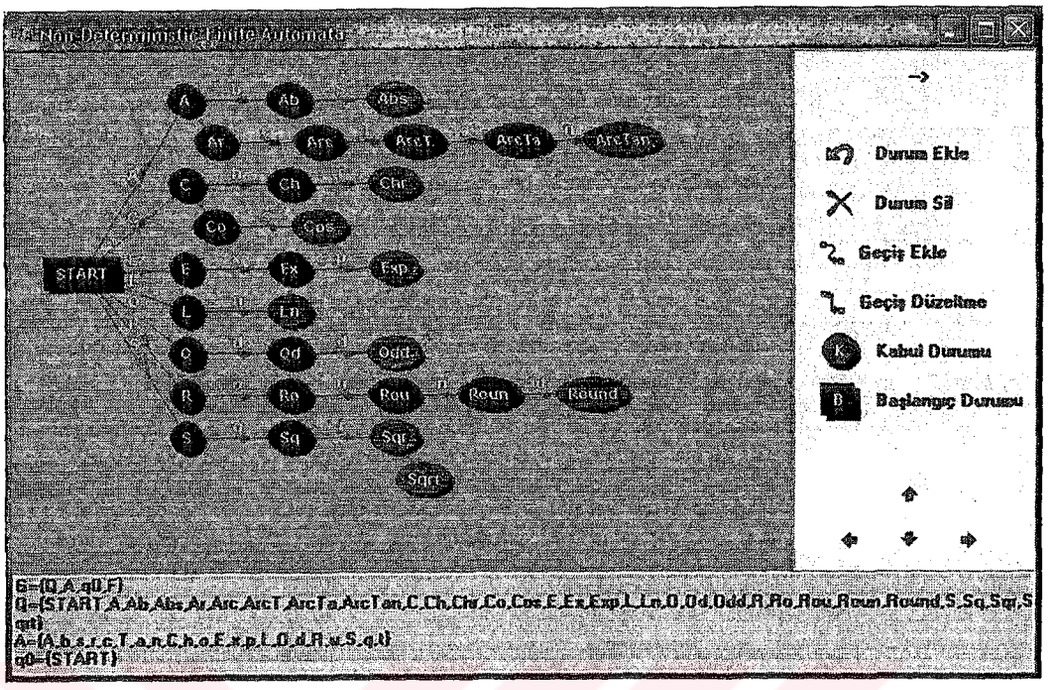
Şekil 6.5. Sözcük Analizi – Operators Penceresi



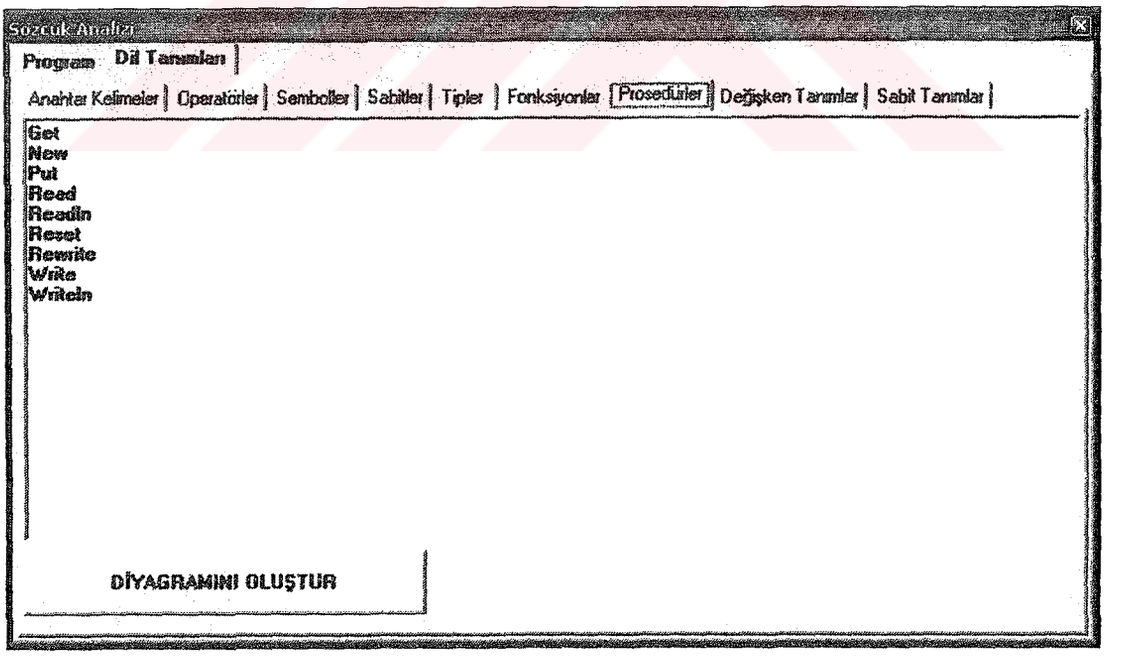
Şekil 6.6. NFA Penceresinde Operatörlerin tanımlanması



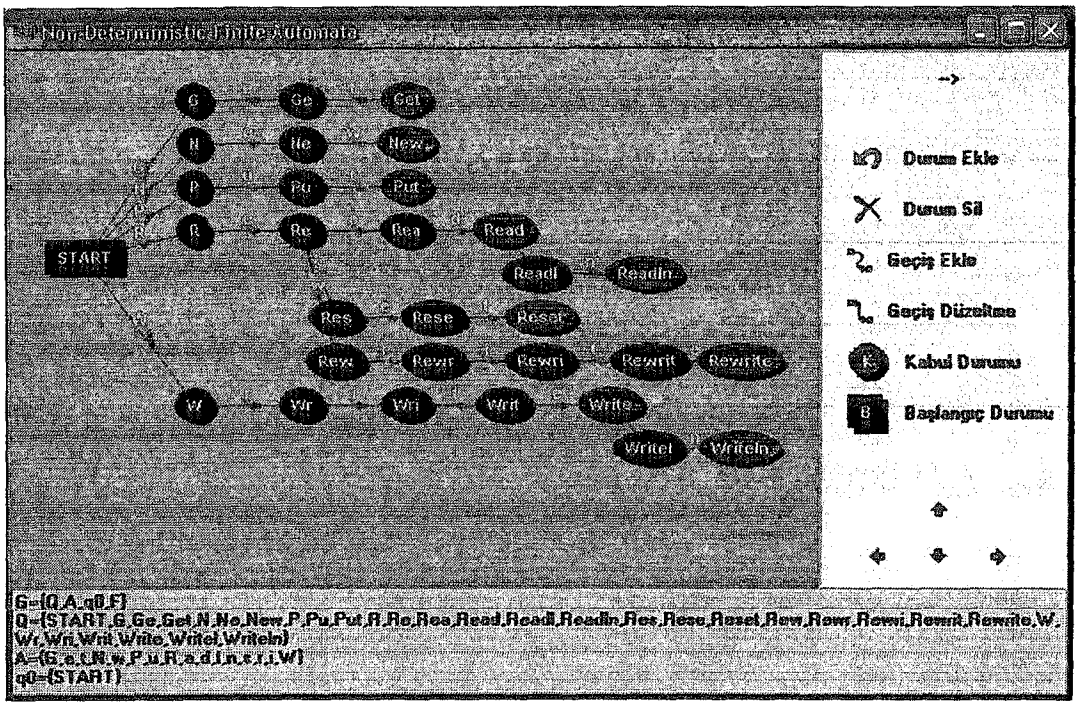
Şekil 6.7. Sözcük Analizi – Functions Penceresi



Şekil 6.8. NFA Penceresinde Fonksiyonların tanımlanması



Şekil 6.9. Sözcük Analizi – Procedures Penceresi

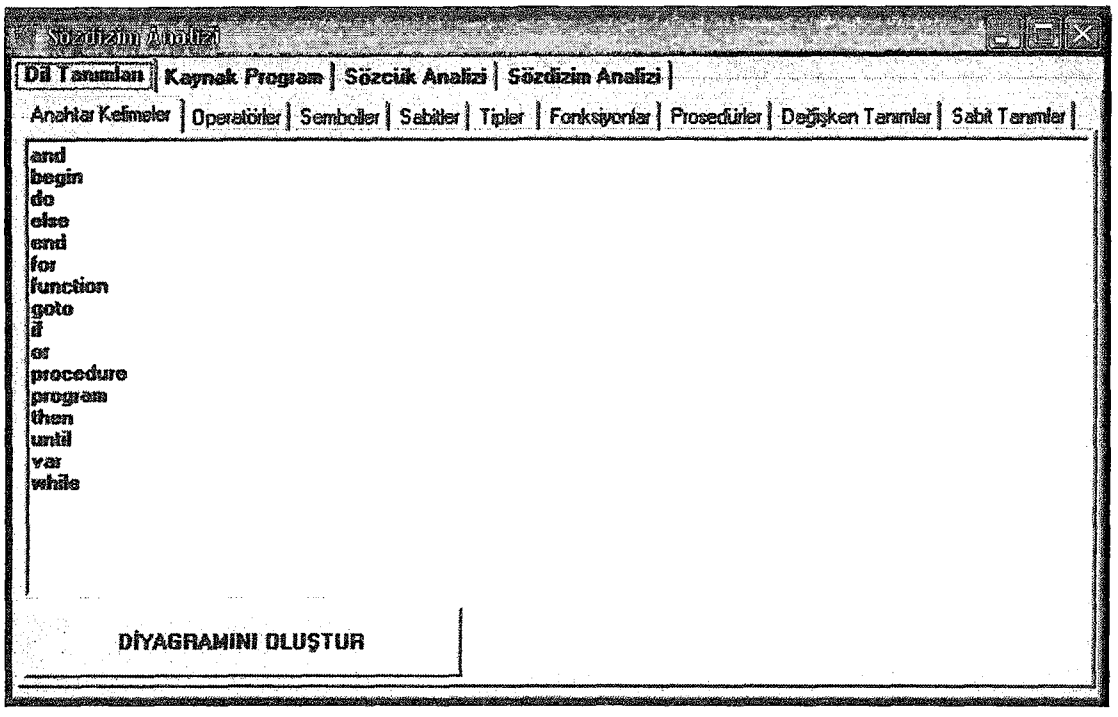


Şekil 6.10. NFA Penceresinde Prosedürler tanımlanması

6.1.2. Sözdizim Analizi

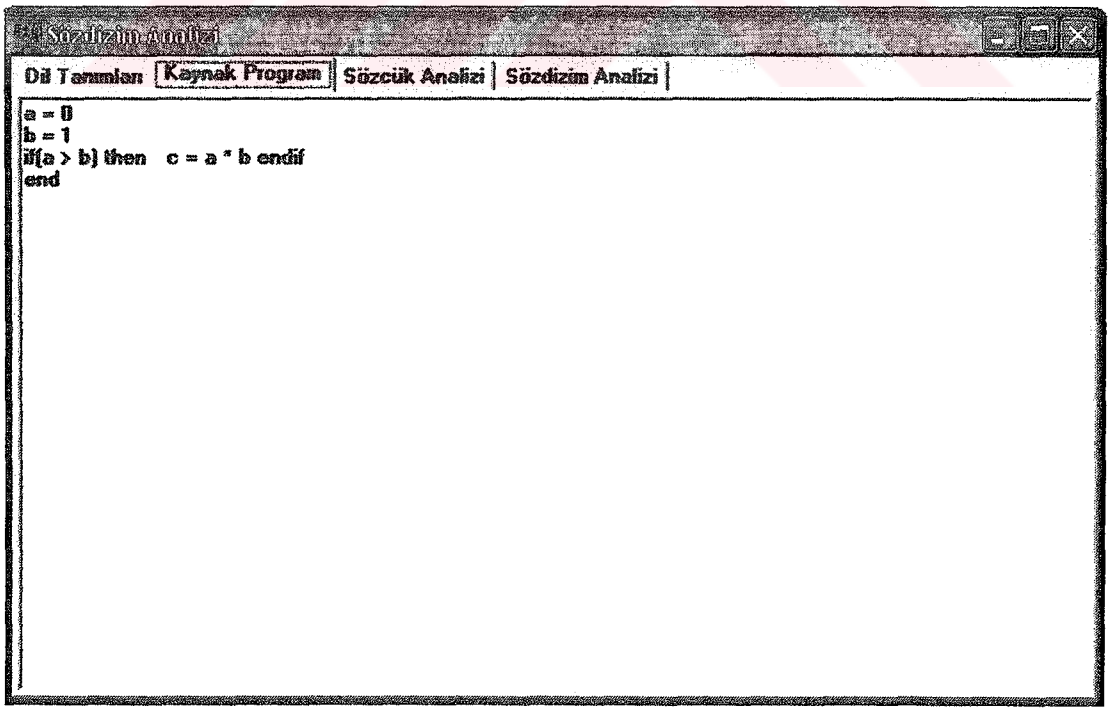
Bir dilin en temel bileşeni alfabesinden aldığı sembollerdir. Bu semboller birleşerek tanımlanan dil için geçerli ve anlamlı kelimeleri meydana getirir. Yine bu kelimeler birleşerek tanımlanan dil için geçerli ve anlamlı cümleler meydana getirirler.

Aşağıda Sözdizim Analiz Penceresine ait görüntü verilmiştir. Kullanıcı bir dile ait sözdizim kurallarının analizi için sırası ile Dilin Tanımları, Kaynak Program, Sözcük Analizi ve son olarak da Sözdizim Analizini kullanması gerekir.



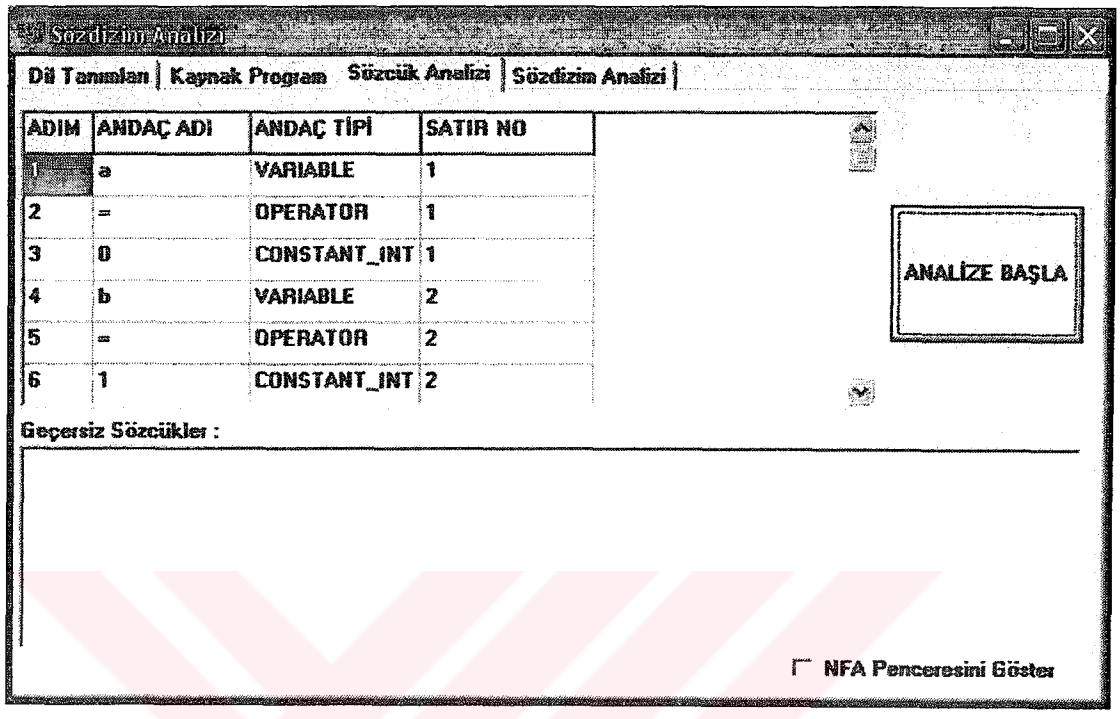
Şekil 6.11. Sözdizim Analizinde Dilin Tanımları

Sözdizim Analizin yapılması için gerekli dili ait tanımların kullanıcı tarafından yukarıdaki gibi girilmesi gerekir.



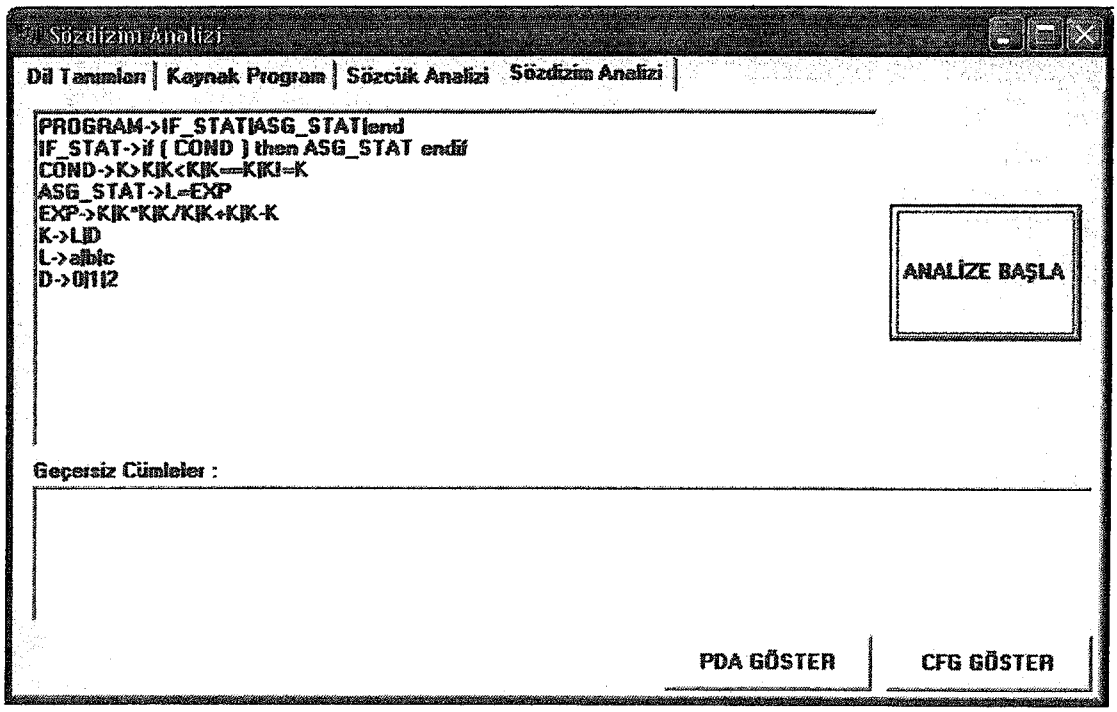
Şekil 6.12. Sözdizim Analizinde Kaynak Program

Sözdizim Analizi yapılacak Kaynak Program yukarıdaki gibi kullanıcı tarafından girilir.



Şekil 6.13. Sözdizim Analizinde Kaynak Program

Sözdizim Analizi için gerekli Andaç Tablosunun oluşturulması ve dilin sözcük analizi için yukarıdaki pencere kullanılır. Dile ait olamayn geçersiz sözcükler liste halinde kullanıcıya verilir.



Şekil 6.14. Sözdizim Analizinde Kaynak Program

Kullanıcının girdiği kaynak programın sözdizim analizini gerçekleştirmek için kullanılır. Sözdizim Analizi Sözcük Analiz tarafından üretilen Andaç Tablosunda kullanıcı tarafından tanımlanan CFG kurallarını uygulayarak hareket eder. Dile ait olmayan geçersiz cümleler liste halinde kullanıcıya verilir.

Örnek olarak verilen bir CFG ve giriş bilgisini CFG ve PDA Geçiş Tablosu yardımı ile Sözdiizm Analizini gerçekleştirelim.

$CUMLE \rightarrow ÖZNE\_N1\_Y1$
$CUMLE \rightarrow ÖZNE\_N2\_Y2$
$CUMLE \rightarrow ÖZNE\_N2\_Y1$

$ÖZNE \rightarrow ben$
$N1 \rightarrow çiçek \mid kitap \mid agac$
$N2 \rightarrow elma \mid çilek \mid portakal$
$Y1 \rightarrow gördüm$
$Y2 \rightarrow ye\,\text{dim}$

CFG'si verilsin. Burada dile ait kelimeleri ve cümleleri bulalım,

Dile ait kelimeler;

- ben
- çiçek
- kitap
- ağaç
- elma
- çilek
- portakal
- gördüm
- yedim

Dile ait cümleleri bulalım;

$CUMLE \rightarrow \ddot{O}ZNE\_N1Y1$ üretimden faydalanarak;

- ben çiçek gördüm.
- ben kitap gördüm.
- ben ağaç gördüm.

$CUMLE \rightarrow \ddot{O}ZNE\_N2Y2$ üretimden faydalanarak

- ben elma yedim.
- ben çilek yedim.
- ben postakal yedim.

$CUMLE \rightarrow \ddot{O}ZNE\_N2Y1$ üretimden faydalanarak

- ben elma gördüm.
- ben çilek gördüm.
- ben portakal gördüm.

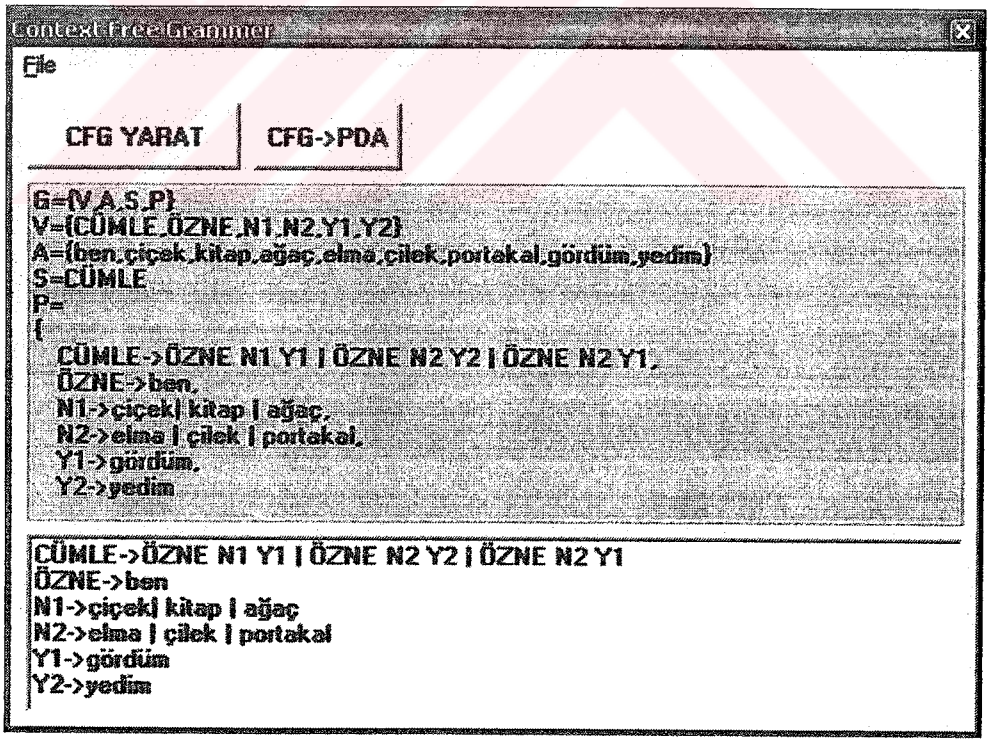
Verilen CFG'ye göre aşağıdaki cümleler dilimiz tarafından kabul edilmemektedir.

- ben çiçek yedim.
- ben kitap yedim.
- ben ağaç yedim.

İşte CFG bu aşamada dilimize ait olan ve olmayan yapıları tanımlamada kullanılır. Bize verilen CFG'de yukarıda dilimiz tarafından kabul edilmeyen üretimler engellenmiştir.

Şimdi yukarıdaki örneği program aracılığı ile çözelim.

Bunun için ilk önce dilimizin sözdizim tanımını sağlayan CFG'yi aşağıdaki gibi oluşturmamız gerekiyor. Ardından "CFG YARAT" düğmesi ile verilen CFG'de yer alan Terminallar, Non-Terminaller, Başlangıç sembolü ve Üretimler program tarafından bulunur.



Şekil 6.15. Örnek CFG

Yukarıdaki pencereden görüldüğü gibi verilen CFG'ye ait bilgiler;

V = {CÜMLE, ÖZNE, N1, N2, Y1, Y2} : Değişkenler kümesi

Σ = {ben, çiçek, kitap, agac, elma, çilek, portakal, gördüm, yedim} : Terminaller

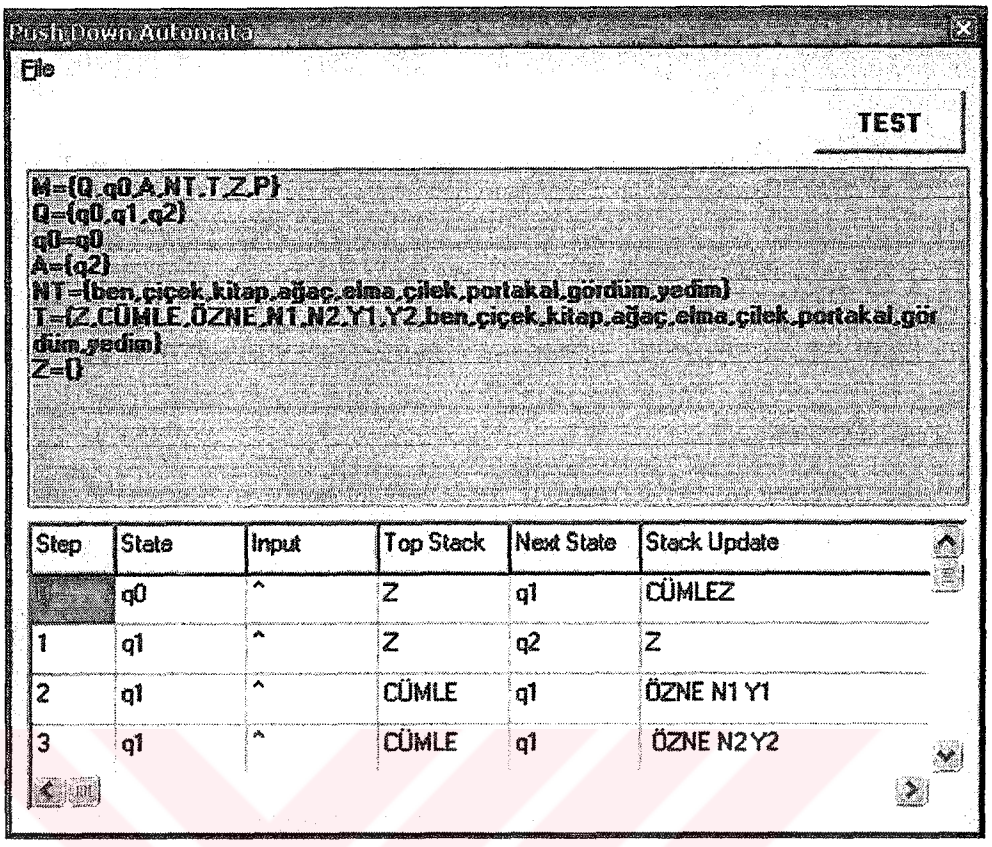
S = {CÜMLE} : Başlangıç sembolü veya değişkeni

P = {
CÜMLE- > ÖZNE N1 Y1 | ÖZNE N2 Y2 | ÖZNE N2 Y1,
ÖZNE- > ben,
N1- > çiçek | kitap | ağğaç
N2- > elma | çilek | portakal,
Y1- > gördüm,
Y2- > yedim }

olarak elde edilir.

Daha sonra "CFG->PDA" düğmesi ile elde edilen CFG test için CFG'nin otomatı olan PDA'ya dönüştürülür. Ve program tarafından PDA Geçiş Tablosu oluşturulur.

Aşağıda yukarıdaki örneğe ait PDA Penceresi ve PDA Geçiş Tablosu yer almaktadır. Burada CFG'den gelen bilgiler kullanılarak PDA'ya ait kümeler bulunur.

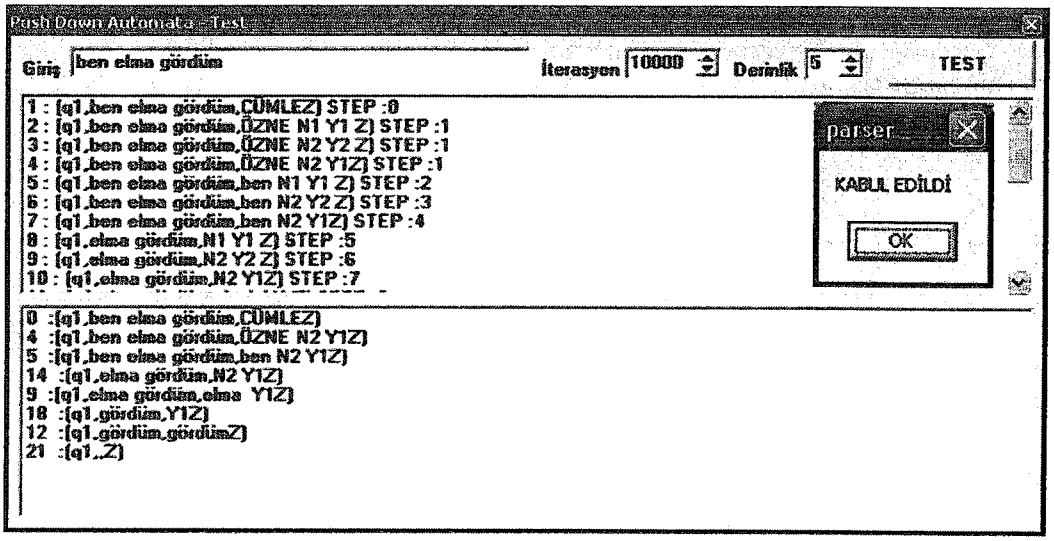


Şekil 6.16. Örnek PDA

PDA Geçiş Tablosunda mevcut durum, giriş bilgisi yığının tepesi bir sonraki surum ve yeni yığın bilgisi bilgileri yer almaktadır. Bu bilgiler kullanılarak o andaki yığın, durum ve giriş bilgisine bağlı olarak bir sonraki durum ve yığın değişimi belirlenmektedir.

Eğer CFG'de belirsizlik durumu varsa yani ayna anda bir Terminal için birden fazla üretim varsa bu PDA Geçiş Tablosuna farklı durumlar olarak eklenir ve incelenir.

Artık girilen CFG'ye ait olan PDA'yı elde etmiş bulunmaktayız. Şimdi bu PDA üzerinde verilen bir giriş bilgisinin dilimiz tarafından geçerli olup olmadığını kontrol edebiliriz. Bunun için "TEST" düğmesi kullanılır ve aşağıdaki gibi bir görüntü elde edilir.



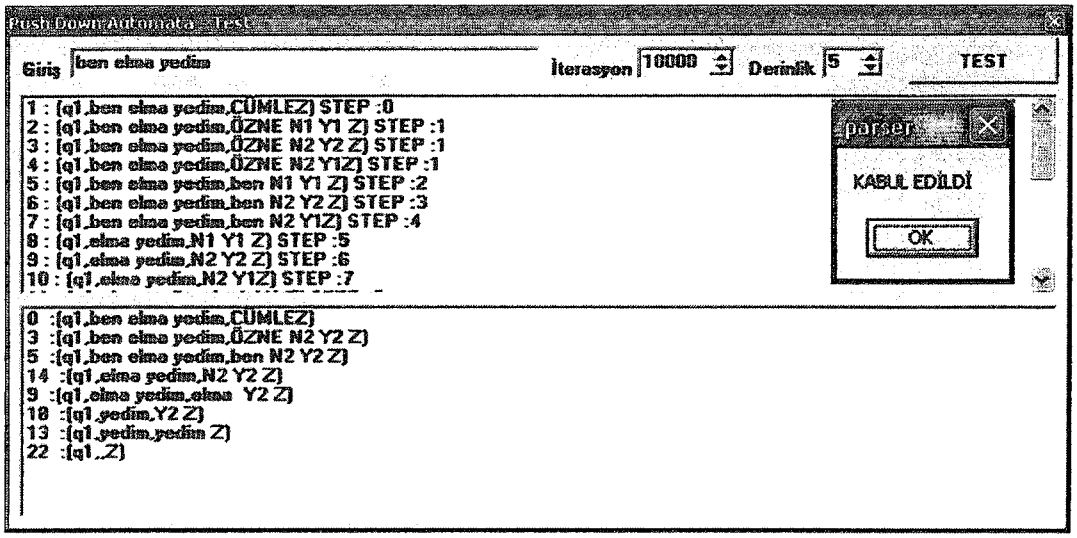
Şekil 6.17. Örnek PDA Test

Yukarıda "Giriş" bilgisi olarak "ben elma yedim" katarı verilmiştir. Giriş yapıldıktan sonra "TEST" düğmesine bastığımızda program bir önceki Pencerede yer alan "PDA Geçiş Tablosundan" yararlanarak verilen giriş bilgisinin geçerli olup olmadığını kontrol eder.

Yukarıdaki örnek de "ben elma yedim" girişi dilimiz için geçerli bir katar olduğundan programın sonucu da "KABUL EDİLDİ" olarak elde ediliyor.

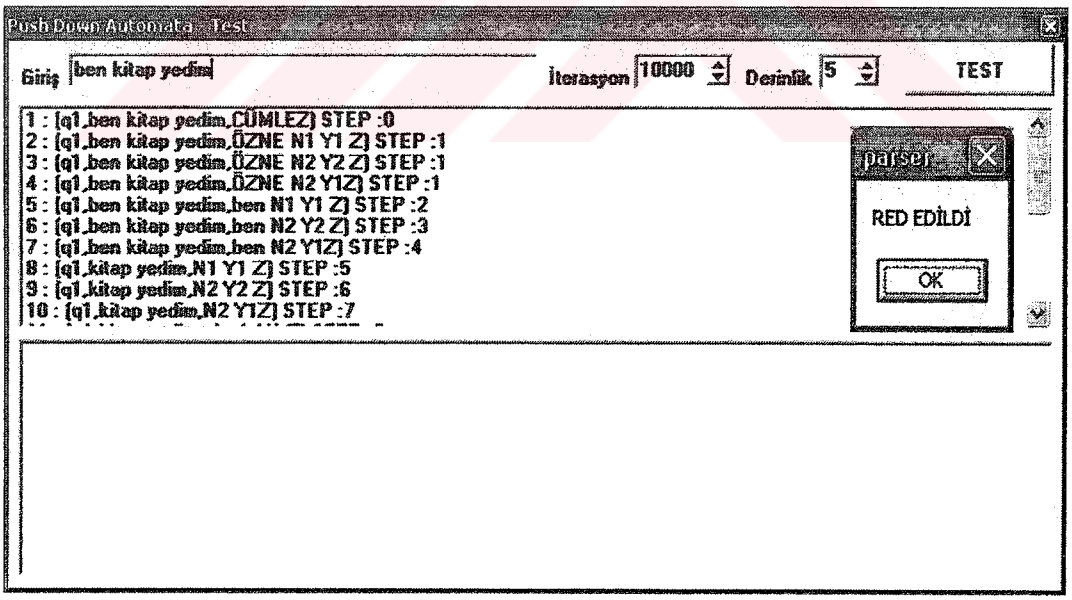
Pencerede iki sonuç kutusu yer almaktadır. Bunlardan üsttekinde o ana kadar bulunan CFG'de tanımlı olan olası durumlar sıralanmaktadır. Alttaki kutuda ise eğer katar kabul edilmiş ise PDA Geçiş Tablosundaki hareket bilgisi yer almaktadır.

Giriş bilgisi olarak "ben elma yedim" katarını girdiğimizde aşağıdaki gibi bir görüntü elde ederiz. "ben elma yedim" katarı verilen CFG dolayısı ile dilimiz tarafından geçerli bir katardır. Bu yüzden programın da cevabı "KABUL EDİLDİ" olarak elde ediliyor.



Şekil 6.18. Örnek PDA Test

Giriş bilgisi olarak "ben kitap yedim" katarını girdiğimizde aşağıdaki gibi bir görüntü elde ederiz. "ben kitap yedim" katarı verilen CFG dolayısı ile dilimiz tarafından geçerli bir katardır değildir. Bu yüzden programın da cevabı "RED EDİLDİ" olarak elde ediliyor.



Şekil 6.19. Örnek PDA Test

6.2. Programın Gerçeklenmesi

Program Borland C++ Builder 6.0 programlama dili kullanılarak gerçeklenmiştir. Programda kullanılan tüm yapılar C++ class yapısı ile tanımlanmıştır. Programda kullanılan temel Class yapıları aşağıdaki gibidir.

- ia_NFA
- ia_CFG
- ia_PDA

6.3. Program Pencereleri

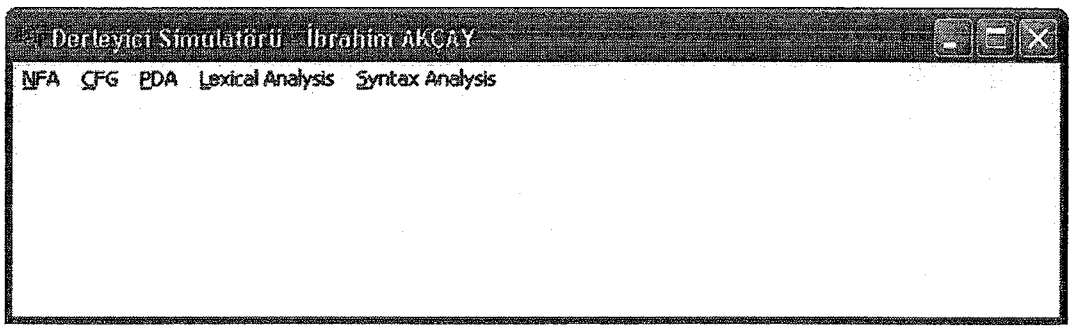
Programda Sözcük Analizi işlemlerinin gerçeklenmesi amacı ile Non-Deterministic Finite Automata ve Sözdizim Analiz işlemleri için Context-Free Gramer ve otomatı olarak Push-Down Automata pencereleri bulunmaktadır.

6.3.1. Ana Pencere – Derleyici Simülatörü

Aşağıda programın ana ekranına ait görüntü yer almaktadır. Programda

- NFA
- CFG
- PDA
- Sözcük Analizi
- Sözdizim Analizi

menüleri ile istenen modül kullanılabilmektedir.



Şekil 6.20. Ana Pencere – Derleyici Simülatörü

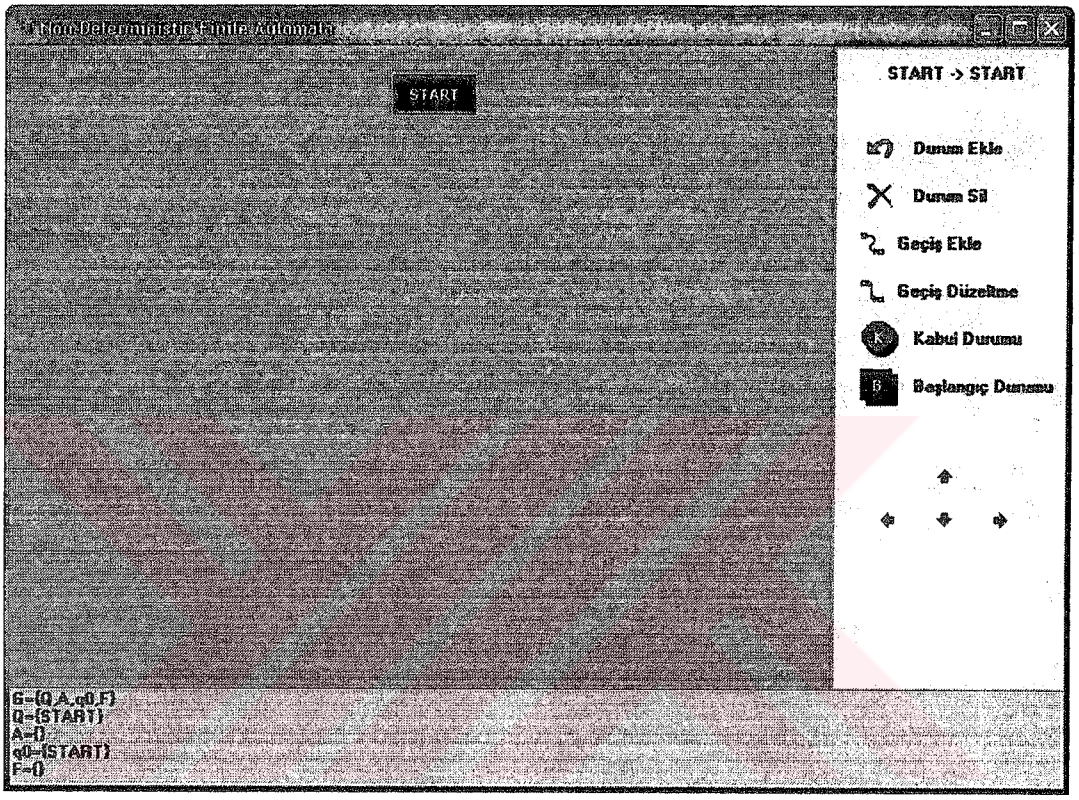
6.3.1. Non-Deterministic Finite Automata

6.3.1.1. Class Yapısı

```
class ia_NFA
{
    ia_NFA();       // Constructor
    ~ia_NFA(){};    // Deconstructor
public :
        vector<ia_NFA_StateTable>       Q;  // Durumlar Kümesi
        vector<string>                  A;  // Alfabe
        string                          q0; // Başlangıç durumu
        vector<string>                  F;  // Kabul durumları kümesi
        vector<ia_NFA_TransitionTable>  T;  // Geçiş Fonksiyonu
// Methods
        void            Clear();
        int             AddState(string newState);
        void            AddAlphabet(string newAlphabet);
        void            AddTransition(int index,string sName,string sTran);
        void            AddTransition(int index,int index1,string sTran);
        void            AddFinalState(string fName);
        bool            Test(string input,TListBox *lists);
        vector<string>  NextTransition(vector<string> states,string input);
        bool            IsFinalState(string state);
        void            CreateNFAForDouble();
        void            CreateNFAForInteger();
        void            CreateNFAForChar();
        void            CreateNFAForString();
        void            CreateNFAForVariable();
        void            AddLetterInAlphabet();
        void            AddDigitInAlphabet();
};
```

6.3.1.2. Pencereler

Non-Deterministic Finite Automata Penceresinde bir NFA makinesi tanımlanabilir ve bu makine üzerinde verilen bir giriş bilgisi kontrol edilebilir. NFA Penceresinin görünümü aşağıda verilmiştir.



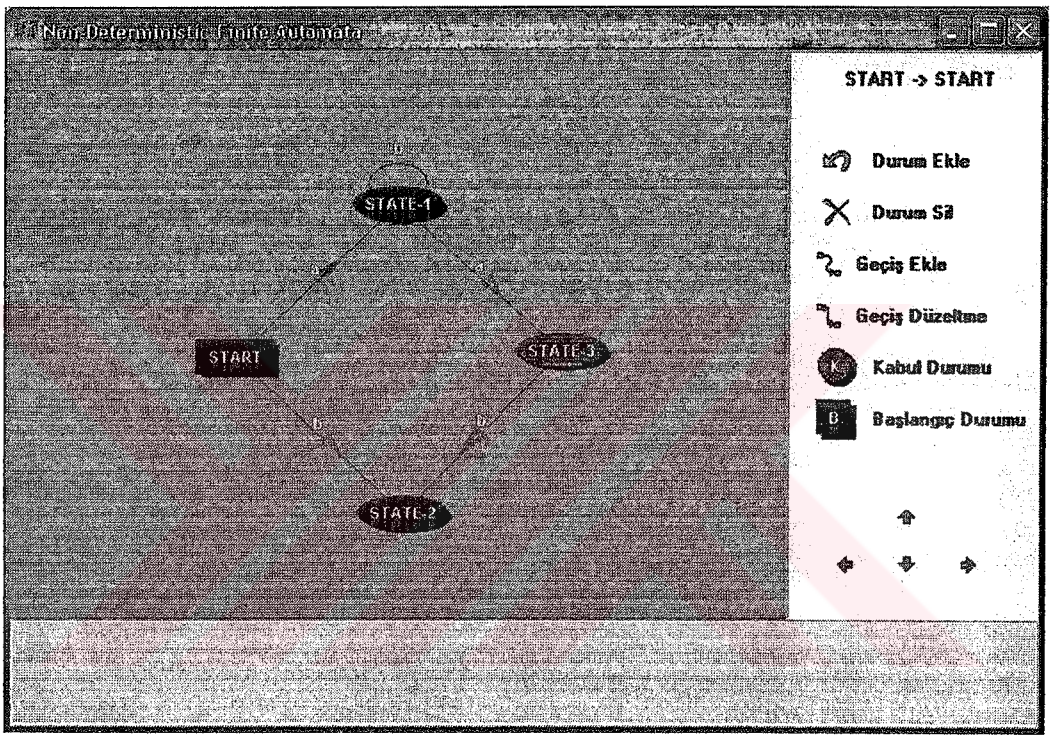
Şekil 6.21. Non-Deterministic Finite Automata Penceresi

NFA tanımlama

Herhangi bir NFA makinesinin program tarafından tanımlanabilmesi için ilk önce NFA diyagramına ait durumlar "Durum Ekle" düğmesi ile sırası ile yaratılır. Ardından bu durumlara ait geçişler ilgili durumları tıkladıktan sonra "Geçiş Ekle" düğmesi tarafından oluşturulur. Makineye ait başlangıç durumu "Başlangıç Durumu" düğmesi tarafından kabul durumu ise "Kabul Durumu" düğmesi tarafından oluşturulur.

Herhangi iki durum arasındaki geçiş ile ilgili bir değişiklik yapmak istenir ise ilgili durumlar seçildikte sonra "Geçiş Düzeltme" düğmesi ile istenen değişiklik yapılabilir.

Oluşturulan diyagramın çeşitli yönlere hareketini sağlamak amacı ile programa "Hareket Düğmeleri" eklenmiştir. Ayrıca oluşturulan diyagram üzerinde verilen bir giriş bilgisini kontrol etmek amacı ile "TEST" düğmesi yer almaktadır.



Şekil 6.22. NFA Penceresinde diyagram oluşturma

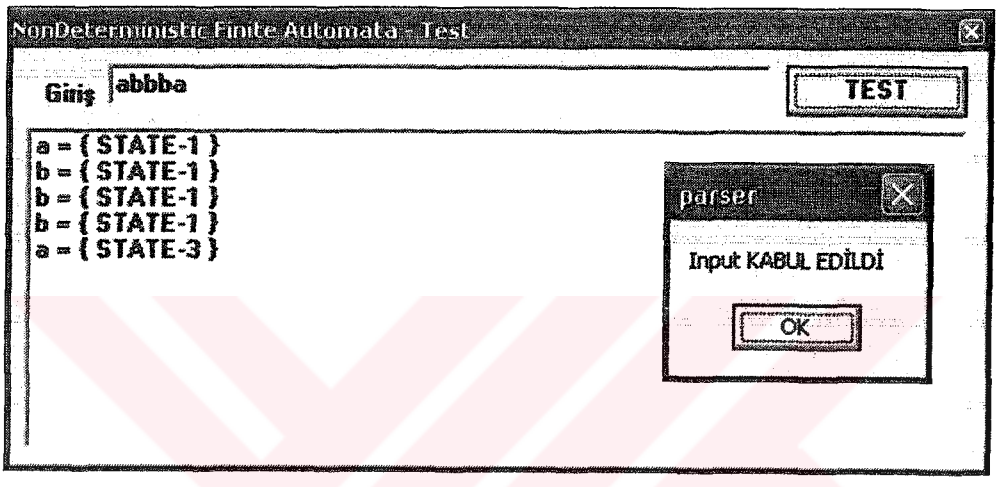
NFA'da test

NFA penceresinde oluşturulan diyagram üzerinde verilen bir giriş bilgisinin geçerliliğini yani makine tarafından kabul edilip edilmediğini kontrol etmek için kullanılır.

Örnek olarak yukarıda NFA penceresinde oluşturduğumuz diyagram üzerinde "abbba" girişini kontrol edelim.

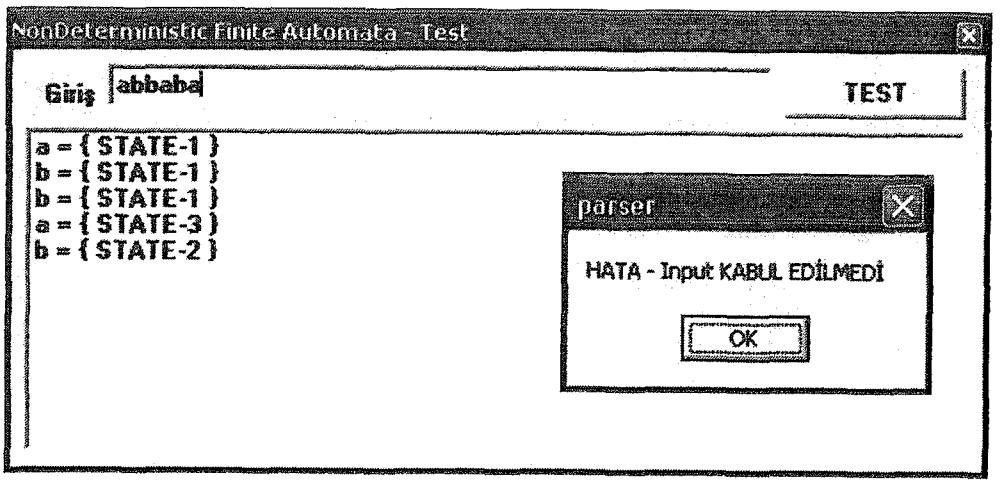
Diyagramdan da anlaşılacağı üzere makinemiz ab^*a regüler ifadesini sağlayan tüm katarları içeren bir dili tanımlamaktadır.

Aşağıda NFA-Test Penceresinde "abbba" giriş bilgisinin kontrol edilmiş görüntüsü yer almaktadır. Verilen giriş bilgisine göre sırası ile hangi durumlara geçiş yapıldığı liste halinde verilmektedir. Giriş bilgimiz olan "abbba" makine tarafından kabul edilmiştir.



Şekil 6.23. NFA Penceresinde verilen bir katarın testi

Şimdi aynı diyagramı "abbaba" giriş bilgisi ile kontrol edelim.



Şekil 6.24. NFA Penceresinde verilen bir katarın testi

6.3.2. Context-Free Gramer

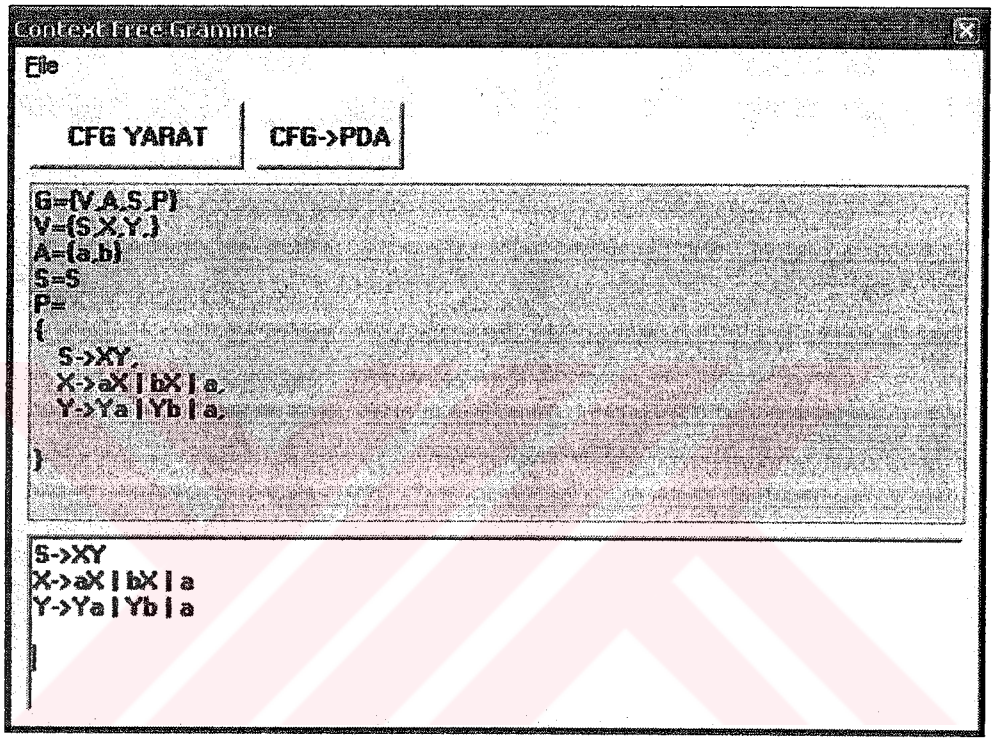
6.3.2.1. Class Yapısı

```
class ia_CFGProductionTable
{
  public:
    string          V;
    vector<string>  production;
};

class ia_CFG
{
  public :
    vector<string>                  V;      // Terminaller kümesi
    vector<string>                  A;      // Non-Terminaller kümesi
    string                          S;      // Başlangıç Non-Terminali
    vector<string>                  P;      // Üretimler
    vector<ia_CFGProductionTable>   PTable; // Üretim Tablosu
    // Methods
    void  Clear();                  // CFG'yi temizler
    void  AddP(string newP);        // Üretim ekleme
    void  AddPT(string newPT);      // Üretim ekleme
    void  AddV(string newV);        // Terminal ekleme
    void  AddA(string newA);        // Non-Terminal ekleme
    void  FindV();                  // Üretimler içinde Terminalleri bulma
    void  FindA();                  // Üretimler içinde Non-Terminalleri
bulma
};
```

6.3.2.2. Pencereler

Context-Free Gramer penceresi bir CFG tanımlamayı ve verilen bu CFG'yi PDA'ya dönüştürerek istenen giriş bilgisi ile kontrol etmeyi sağlar.

Aşağıda Context-Free Grammer Penceresine ait görüntü yer almaktadır.



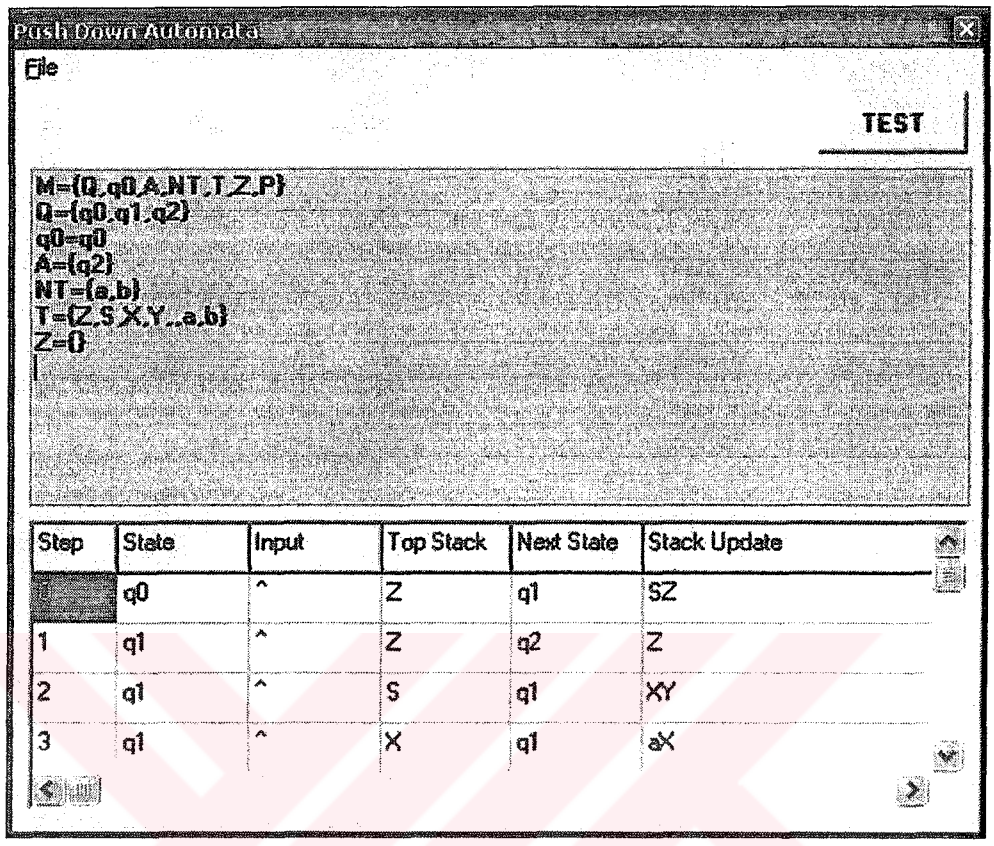
Şekil 6.25. Context-Free Gramer Penceresi

Kullanıcı tarafından CFG'ye ait üretimler pencerenin altında bulunan girişten girildikten sonra "CFG YARAT" düğmesi ile bu üretimlerden faydalanarak CFG'ye ait olan Terminaller, Non-Teminaller ve Başlangıç sembolü oluşturulur.

Oluşturulan bu CFG üzerinde herhangi bir giriş bilgisinin kontrolü için CFG'nin PDA'ya dönüştürülmesi gerekiyor. Çünkü CFG otomat olarak PDA'yı kullanır.

Bu işlem pencere üzerindeki "CFG->PDA" düğmesi ile gerçekleştirilebilir.

Aşağıda yukarıda verilen CFG'ye ait PDA görüntüsü yer almaktadır.



Şekil 6.26. Push-Down Automata Penceresi

6.3.3. Push-Down Automata

6.3.3.1. Class yapısı

```
class ia_PDATransitionTable
{
  public :
    int      step;
    int      index;
    int      priorIndex;
    string   state;
    string   input;
    string   topStack;
    string   nextState;
```

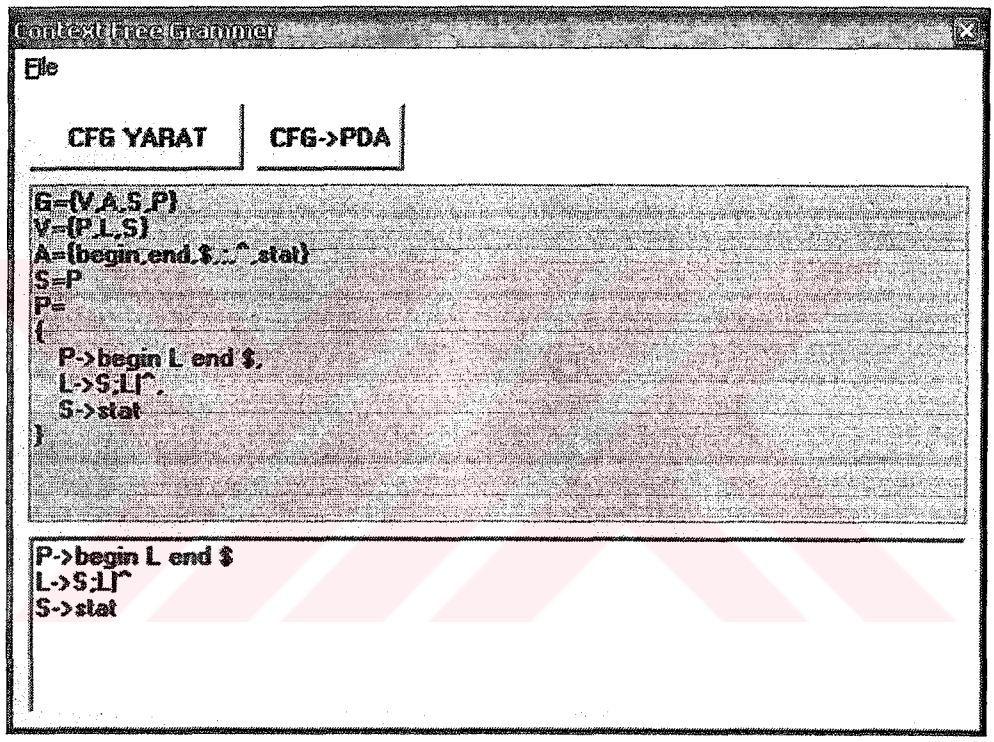
```
    string   stackUpdate;
};

class ia_PDA
{
  public  :
    vector<string>                  Q;      // Durumlar kümesi
    string                          q0;     // Başlangıç durumu
    vector<string>                  A;      // Kabul Durumlari kümesi
    vector<string>                  NT;     // Giriş Alfabesi
    vector<string>                  T;      // Yığın Alfabesi
    vector<string>                  Z;      // Başlangıç Yığın Sembolü
    vector<ia_PDATransitionTable>   P;      // Üretimler
    // Methods
    void    Clear();
    void    PrintTransitionTable(TStringGrid *table);
    void    Print(TMemo *printMemo);
    void    AddQ(string newQ);
    void    Addq0(string newq0);
    void    AddA(string newA);
    void    AddNT(string newNT);
    void    AddT(string newT);
    void    AddZ(string newZ);
    void    AddP(ia_PDATransitionTable newTransTable);
    bool    Test(string input,int sDepth,int maxIteration);
    string  GetNextInput(string inputs);
    string  GetNextStack(string stack);
};
```

6.3.3.2. Pencereler

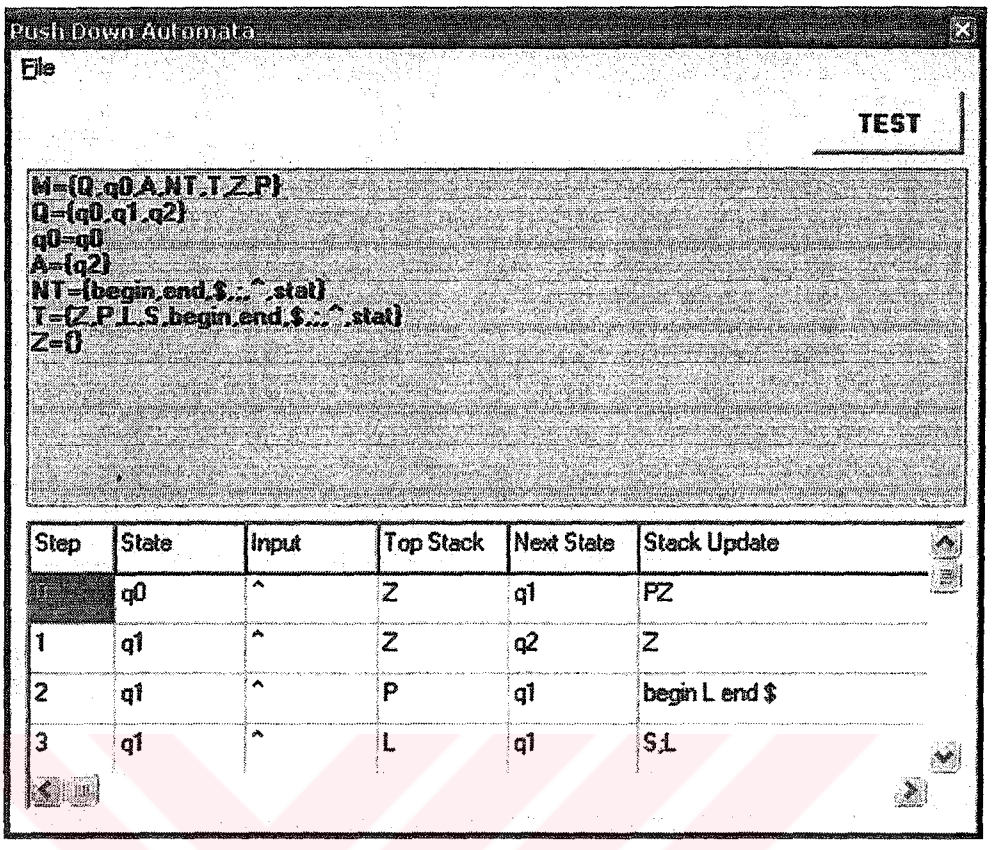
CFG tanımlama

PDA genel olarak verilen bir CFG'nin otomatı olarak kullanılır. Programda da verilen CFG'nin kontrol edilmesini sağlar.

Aşağıda bir CFG ve ondan elde edilmiş PDA penceresine ait bir görüntü yer almaktadır.

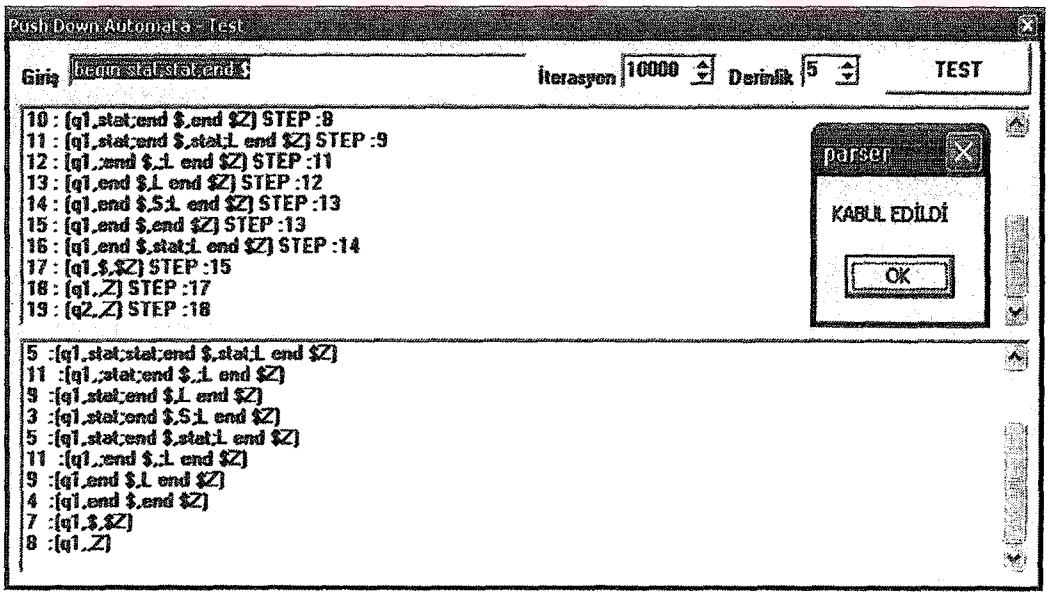


Şekil 6.27. CFG'de verilen bir makinenin tanımlanması

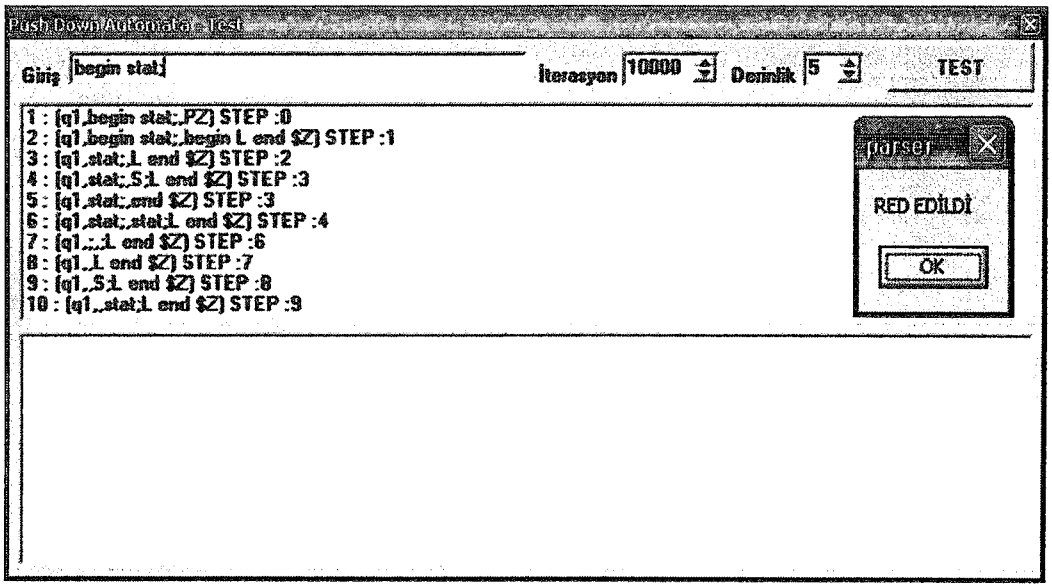


Şekil 6.28. CFG'de tanımlı bir makinenin PDA'ya aktarılması

PDA'da test



Şekil 6.29. PDA'da verilen bir katarın testi



Şekil 6.30. PDA'da verilen bir katarın testi

BÖLÜM 7. SONUÇLAR VE ÖNERİLER

Program kullanıcı tarafından bir dilin tanımlanmasını ve bu dile ait ve yine kullanıcı tarafından tanımlanan sözcük ve sözdizim yapılarını kullanarak verilen bir kaynak program üzerinde sözcük ve sözdizim analizini gerçekleştirir.

Yukarıda verilen veri yapısından anlaşılacağı gibi modüller class kullanılarak yaratılmıştır. Program arayüzü kullanıcı açısından çok uygundur. Kullanıcının hızlı bir şekilde verileri girmesi ve sonuçları adım adım görsel olarak ekrandan izlemesi mümkündür. Kullanıcı Sözcüksel ve Sözdizim Analiz için gerekli tanımları ilgili pencerelerde gerçekleştirip programın çıktılarını ayrıntılı olarak izleyebilir. Bu yönden büyük bir kolaylık sağlamaktadır. Çünkü dilin kendisi ve yapısı direkt olarak kullanıcı tarafından oluşturulabilmektedir. Ayrıca herhangi bir yapı bu dilin tanımından bağımsız olarak tanımlanabilir ve üzerinde istenen işlemler yapılabilir.

NFA, CFG, PDA gibi konular üzerinde görsel uygulamaların kullanıcı ile etkileşimli gerçekleştirildiği bir çalışma olması bakımından eğitim amaçlı kullanılabilir.

Program genel amaçlı bir dilin tanımını ve bu dil üzerinde sözcük analizi ve sözdizim analizini gerçeklemektedir. Yani program giriş olarak dilin tanımlarını ve kaynak programı istemektedir. Çıkış olarak sözcük analizi ve sözdizim analiz sonuçları verilmektedir. Bu yapısını herhangi bir uygulamada kullanımını sağlamak amacı ile giriş olarak istenen dilin kuralları ve kaynak programı belirtilen dosyalardan alarak sonucu yine belirtilen bir dosyaya aktaran bir yapıya dönüştürülmesi gerekmektedir.

Ayrıca kullanıcı tarafından verilen CFG'ye bağlı olarak kullanılan algoritmanın verilen kaynak programı analizi uzun sürebilir. Bu nedenle algoritmaya iterasyon sayısı ve arama derinliği olmak üzere iki parametre girilmektedir. Bu şekilde CFG'de

var olan belirsizlikten kaynaklanan sonsuz döngü ve sonuca ulaşamama engellenmiştir. Bu noktada var olan algoritmaya belirsizliği ortadan kaldırma yöntemleri eklenebilir.

KAYNAKLAR

[1] Reinhard W., Dieter M., Compiler Design, Addison-Wesley

[2] Daniel I.A.C., Introduction To Computer Theory, Second Edition, Jhon Wiley&Sons, Inc 1997

[3] Jhon C. M., Introduction To Language And Theory Of Computation, International Editions 1997

[4] Jhon A.D., Albert D.O., Lawrence E.S., Charles V.E., Discrete Mathematics, Addison-Wesley

[5] Alblas, H. and Nymeyer, Practice and Principles of Compiler Building with C, Prentice-Hall 1996

[6] Andrews, G.R. and Schneider, F.B. Concepts and notation for concurrent programming,ACM Computing Surveys

[7] Backhouse, R.C. Syntax of Programming Languages: Theory and Practice, Prentice-Hall,Hemel Hempstead, England.

[8] Ben-Ari, M. Principles of Concurrent Programming, Prentice-Hall, Englewood Cliffs, NJ. 1982

[9] Holub, A.I. Compiler Design in C, Prentice-Hall, Englewood Cliffs, NJ. Lee, J.A.N.

[10] Tremblay, J.P. and Sorenson, P.G. Theory and Practice of Compiler Writing, McGraw-Hill,New York. 1985

EKLER

Bitirme Tez Çalışmasına ait hazırlamış olduğum program kodları ve çalıştırılabilir program, CD eki olarak verilmiştir. Program Borland C++ Builder 6.0 Programlama Dilinde gerçeklenmiştir. Tüm modüller ve ilgili dosyalar diskette bulunmaktadır. Bu program ile Bitirme Tez Çalışmasında yukarıda ele alınan tüm konulara ilişkin kendi oluşturduğum algoritma ile probleme ilişkin çözümler verilmektedir.

ÖZGEÇMİŞ

1976 yılında Gaziantep'in Nizip ilçesinde doğdum. İlköğretimimi Hatay Kuşalanı İlköğretim Okulunda, Ortaöğretimimi Gaziantep İsmet Paşa Lisesinin Orta kısmında ve Lise eğitimimi de Gaziantep M.A. Ersoy Teknik Lisesi Bilgisayar Donanımı Bölümünde tamamladım. 200-2001 öğretim yılında Sakarya Üniversitesi Mühendislik Fakültesi Bilgisayar Bölümünden mezun oldum.