

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

**BİÇİMSEL DİLLERDEN ENDÜSTRİYEL
İŞLEMCİLERE OTOMATİK KOD ÜRETME:
OTOMAT YAKLAŞIMI**

YÜKSEK LİSANS TEZİ

Müh. Özde TİRYAKİ

**Anabilim Dalı: Kontrol ve Bilgisayar Mühendisliği
Programı: Kontrol ve Bilgisayar Mühendisliği**

OCAK 2007

**BİÇİMSEL DİLLERDEN ENDÜSTRİYEL
İŞLEMCİLERE OTOMATİK KOD ÜRETME:
OTOMAT YAKLAŞIMI**

YÜKSEK LİSANS TEZİ

Müh. Özde TİRYAKİ

504991082

Tezin Enstitüye Verildiği Tarih : 25 Aralık 2006

Tezin Savunulduğu Tarih : 29 Ocak 2007

Tez Danışmanı Prof.Dr. Leyla GÖREN

Diğer Jüri Üyeleri Doç. Dr. Salman Kurtulan (İ.T.Ü.)

Yrd. Doç. Dr. Osman Kaan Erol (İ.T.Ü.)

OCAK 2007

ÖNSÖZ

Yapmış olduğum bu çalışmada benden yardımlarını esirgemeyen Prof. Dr. Leyla Gören'e, Doç. Dr. Salman Kurtulan'a, birlikte çalıştığım arkadaşım Anıl Şahin Zimba'ya ve aileme teşekkürü bir borç bilirim.

Ocak 2007

Özde Tiryaki

İÇİNDEKİLER

KISALTMALAR	v
ŞEKİL LİSTESİ	vi
SEMBOL LİSTESİ	vii
ÖZET	viii
SUMMARY	ix
1. GİRİŞ	1
1.1 Sistem ve Model Kavramı	2
1.2 Sistemlerin Sınıflandırılması ve Zaman Kavramı	3
1.3 Durum Kavramı ve Durum Uzayı Modeli	4
1.4 Geribesleme Kavramı	6
1.5 Ayırık Olay Sistemleri	6
1.5.1 Zaman denetimli ve olay denetimli sistemler	7
1.5.2 Ayırık olay sistemlerinin karakteristik özellikleri	9
1.5.3 Ayırık olaylı sistemlerde üç seviyede soyutlama	12
1.6 Otomatlar ve Petri Ağları	12
1.7 Tez çalışmasının amacı ve elde edilen sonuçlar	13
2. DİLLER VE OTOMATLAR	14
2.1 Sonlu Durum Sistemler ve Dil Modelleri	14
2.1.1 Diller üzerindeki işlemler	16
2.2 Otomatlar	17
2.3 Otomatlarla Gösterilen Diller	19
2.3.1 Otomatların eşdeğerliği	20
2.3.2 Kilitlenme	21
2.3.3 Belirgin olmayan otomatlar	22
3. OTOMATLAR ÜZERİNDE İŞLEMLER	24
3.1 Tekli İşlemler	24
3.2 Kompozisyon İşlemleri	27
4. SONLU DURUMLU OTOMATLAR	31
4.1 Regüler Dillerin Tanımı ve Özellikleri	31
4.2 Regüler İfadeler	33
5. OTOMATLAR VE PETRİ AĞLARININ KARŞILAŞTIRILMASI	34
5.1 Dilin İfade Edilebilirliği	34
5.2 Modüler Model İnşa Etme	36
5.3 Karar Verilebilirlik	37
6. PROGRAM AÇIKLAMALARI	37
6.1 Borland C++ Builder Programı	39
6.2 PLC SCL Programı	66

6.2.1	Durum geiş diyagramı yöntemi	66
6.2.2	Petri ağı A matrisi yöntemi	71
KAYNAKLAR		77
ÖZGEÇMİŞ		78

KISALTMALAR

DES	: Discrete Event Systems
CVDS	: Count Variable Dynamical System
PLC	: Programmable Logic Controller
SCL	: Structural Code Language
PNL	: Petri Net Language

ŞEKİL LİSTESİ

	<u>Sayfa No</u>
Şekil 1.1 : Durum Uzayı Modeli	4
Şekil 1.2 : Rasgele Yürüyüş	7
Şekil 1.3 : Olay Denetimli Rasgele Yürüyüş	9
Şekil 1.4 : DES ve CVDS Yörüngeleri	11
Şekil 1.5 : İşaret AkışDiyagramı	14
Şekil 2.1 : Durum Geçiş Diyagramı	17
Şekil 2.2 : Örnek 2.2'nin durum geçiş diyagramı	20
Şekil 2.3 : Örnek 2.3'nin durum geçiş diyagramı	20
Şekil 2.4 : Üç eşdeğer otomat	21
Şekil 2.5 : Açmaz ve kısır döngü örnekleri	22
Şekil 2.6 : Belirgin olmayan otomat örneği	23
Şekil 2.7 : Belirgin otomat örneği	24
Şekil 3.1 : G otomatı	27
Şekil 3.2 : (a) CoAc(G), (b) Trim(G), (c) Trim(G)'nin tümleyeni	27
Şekil 3.3 : İki otomatın bağlantısı	28
Şekil 3.4 : Otomatların çarpımı	29
Şekil 3.5 : Şekil 2.1 ve Şekil 2.2'deki otomatların paralel kompozisyonu ...	30
Şekil 4.1 : L dilini işaretleyen otomat	32
Şekil 5.1 : G'deki (x, e, x') üçlüsünün N'deki karşılığı	36
Şekil 5.2 : G otomatının, N Petri ağı eşleniği	36
Şekil 6.1 : Programa İlişkin Akış Diyagramı	38
Şekil 6.2 : Program Arayüzü	39
Şekil 6.3 : Durum Geçiş Diyagramı Yöntemi İşaret Akış Diyagramı	70
Şekil 6.4 : Petri Net A Matrisi Yöntemi İşaret Akış Diyagramı	76

SEMBOL LİSTESİ

G	: Otomat
N	: Petri ağı
X	: Durum kümesi
E	: Olay kümesi
<i>f</i>	: Durum geçiş fonksiyonları
Γ	: Aktif olay kümesi
x_0	: Başlangıç durumu
X_m	: İşaretli durumlar
ε	: Boş kelime
P	: Yer kümesi
T	: Geçiş kümesi
A	: A Petri ağ matrisi
ω	: Ark ağırlık fonksiyonu
<i>l</i>	: Geçişlerin etiketlenme fonksiyonu
L	: Dil
$\mathbb{L}_m$	: İşaretli dil
$\mathbb{L}$	: Üretilen dil
$\mathcal{R}$	: Regüler dil

BİÇİMSEL DİLLERDEN ENDÜSTRİYEL İŞLEMCİLERE OTOMATİK KOD ÜRETME: OTOMAT YAKLAŞIMI

ÖZET

Bu çalışmada, biçimsel işaretli bir dilden endüstriyel bir işlemciye otomatik kod üreten bir program geliştirme amaçlanmıştır. Bu amaç için, önce işaretli dilden üretilen dile geçilmiş ve ilişkin otomatın durum geçiş diyagramı elde edilmiştir. Daha sonra durum geçiş matrisinden kullanılan işlemciye uygun kod üreten yazılım gerçekleştirilmiştir. Bu işlemler, tüm aşamaları görsel olarak sunmaya imkan veren Borland C++ Builder ortamında gerçekleştirilmiştir. Endüstriyel işlemci olarak SIMATIC-300 seçilmiş ve standart bir dil olan SCL dilinde kod üretilmiştir. Benzer yazılımlarda ortaya çıkan ve ilgili literatürde “çığ etkisi” (avalanche effect) olarak adlandırılan problem analiz edilmiş ve çığ etkisinden arındırılmış SCL kodu üretilmiştir. Oluşturulan yazılım çeşitli endüstriyel örnekler üzerinde denenmiş ve başarılı sonuçlar alınmıştır.

GENERATING AUTOMATIC CODE FROM FORMAL LANGUAGES TO INDUSTRIAL PROCESSORS: AUTOMATA APPROACH

SUMMARY

In this study, target is generating automatic code from formal languages to industrial processors. For this purpose, first marked language is converted to generated language and state transition diagram of the related automata is identified. Then, a software which generates automatic code from state transition matrix is developed in a language appropriate for the processor. This software is developed in BORLAND C++ Builder due to its visual properties. For industrial processor, SIMATIC-300 is chosen and code is generated in SCL language which is the standard language of this processor. Avalanche effect problem which can be seen in similar softwares is analyzed and SCL code is improved to eliminate this problem. Generated software is tested in various industrial examples and successful results are achieved.

1. GİRİŞ

Bilgisayar, haberleşme ve sensor teknolojilerindeki hızlı gelişmeler, “yeni” dinamik sistemlerin sayılarında büyük bir artışa sebep olmuştur. Bu “yeni” dinamik sistemlerin bazıları teknolojiktir ve çoğu da çok karmaşıktır. Bilgisayar ve haberleşme ağları, otomatik üretim sistemleri, hava trafik kontrol sistemleri, akıllı ulaşım sistemleri, dağıtılmış yazılım sistemleri bunlardan birkaçıdır. Bu sistemlerdeki “aktivite”lerin önemli bir kısmı, hatta bazılarında tümü, insanlar tarafından tasarlanmış işlemsel kurallara bağlı olarak oluşurlar ve davranırlar. Bu nedenle de, bunların dinamikleri “ayrık olayların” asenkron olarak oluşmasıyla karakterize edilir; bu olayların bazıları kontrollu (klavyenin bir tuşuna basılması) olabilirken bazıları ise kontrolsüz olabilir (bir cihazın aniden bozulması). Bu gerçekler nedeniyle, dinamik sistemlerin bu sınıfı için “ayrık olay sistemler” terimi kullanılır.

Günümüzdeki diferansiyel ve fark denklemleri merkezli matematiksel birikim (ki bunlar sistem ve kontrol mühendisliğinde, ait oldukları doğa yasaları nedeniyle “zaman denetimli” sistemlerin modelleri ve çalışmaları için çok uzun zamandır kullanılmaktadır) ayrık olay sistemleri için uygun değildir. Bu nedenle yeni modelleme yöntemleri, analiz teknikleri, tasarım araçları, test metodları ve sistematik kontrol ve optimizasyon prosedürleri geliştirilmesi gerekir ki, bu “yeni” nesil karmaşık sistemlerin analizi ve tasarımı mümkün olabilsin.

Bu yeni nesil karmaşık sistemlerden biri olan bilgisayarın kendisi bu tür sistemlerin tasarımı, analizi ve kontrolü için yeni tekniklerin ve prosedürlerin geliştirilmesinde önemli bir rol oynamaya başlamıştır. Ayrık olay sistemlerinin sahip olduğu veya olmaya yüz tuttuğu kapasite ne kadar heyecan verici ise sistemlerin karmaşıklığı da bir o kadar düşündürücüdür. Güçlü methodların bulunması sadece tasarım yöntemlerinin geliştirilmesi için değil aynı zamanda hataları önlemek için de gereklidir. Bu derece karmaşık sistemlerde hatalar yıkıcı olabilir.

Tarihsel olarak, bilim adamları ve mühendisler; fizik, kimya, mekanik ve yer çekimi yasaları ile iyi bir şekilde modellenen doğa olayları üzerinde çalışmaya yoğunlaşmışlardır. Bu nedenle parçacıkların ve katı maddelerin yer değiştirmesi, hızı ve ivmesi, sıvı ve gazların basıncı, sıcaklığı ve akış hızları gibi büyüklüklerle

uğraşmaktadır. Bunlar “sürekli değişkenlerdir”, yani değerleri zamana göre sürekli olarak değişir. Bu gerçeğe dayanarak, matematiksel araçlar ve teknikler bu türden sistemlerin modellenmesi, analizi ve kontrolü için geliştirilmişlerdir. Adi ve kısmi diferansiyel denklemler sistem analizi ve kontrolünün temelini oluşturur. Fakat günümüzde giderek artan teknolojik ve bilgisayar bağımlı dünyada iki önemli noktaya dikkat etmek gerekir;

- Karşılaştığımız büyüklüklerin çoğu “ayrıktır” ve tam sayılarla sayılırlar. (Kaç adet uçak kalkmakta, kaç adet telefon görüşmesi aktif halde, vs.)
- Kullandığımız proseslerin devreye girmesi anlık olarak olan “olay”lara bağlıdır. (Klavyenin tuşuna basmak, trafikte yeşil ışığın yanması, vs.) Bu sistemler “olay denetimli” olarak adlandırılırlar.

1.1 Sistem ve Model Kavramı

Tanım 1.1 “Sistem”:

Sistemin literatürdeki üç tanımı şöyledir:

- Karmaşık bir tamlık veya bütünlük oluşturmak üzere doğa ve insan tarafından biraraya getirilen şeylerin toplanması veya birleşmesi (Enc Americana)
- Bir tam birleşme oluşturan elemanların bağımsız bir grubu veya düzgün bir etkileşimi (Webster Dict.)
- Teke tek parçalarla gerçekleştirilmesi mümkün olmayan bir fonksiyonu (amacı) yerine getirmek için birarada davranan elemanların bir birleşimi (IEEE Standart Dict. Of Electric&Electronic Terms)

Bu tanımların iki temel gerçeği vardır:

- Sistem birbiriyle etkileşimli olan elemanlardan oluşur.
- Sistem amaç fonksiyonu yerine getirmek amacıyla biraraya getirilir. Burada biraraya getirilirken ön varsayım bir amacı yerine getirmektir.

Vurgulanması gereken diğer önemli bir nokta da bir sistemin her zaman fiziksel objeler ve doğa yasaları ile biraraya getirilmesinin zorunlu olmadığıdır. Örneğin sistem teori, ekonomik mekanizmaların veya insan davranışının ve toplumsal dinamiklerin tanımlanmasında çok uygun bir çalışma zemini oluşturur.

Sistemlerin analizi, kontrolü ve tasarımı için matematiksel ölçümler ve işlemler yapmak gereklidir bu nedenle bazı tanımlara ihtiyaç duyulur.

Tanım 1.2 “Model”:

Bilimci veya mühendisler sistemlerin büyüklük analizi ile, kontrol ve tasarım için bir takım tekniklerin geliştirilmesi ile ve iyi tanımlanmış ölçütler üzerinden sistem davranışının kapalı ölçümleri ile ilgilenirler. Bu nedenle yukarıda verilen tümüyle niteliksel tanımlar uygun olmaz. Geçerli olan sistemin bir “model”i aranır. Sezgisel olarak, sistemin kendi davranışı ile basit olarak uyuşan bir cihaz (araç) olarak bir model düşünülür. Daha ayrıntılı açıklamak gerekirse, bu davranışı anlamak için bazı matematiksel kavramlar geliştirmeye ihtiyaç duyulur. Kesin olarak söylenebilir ki; bir sistem gerçek bir şeydir (bir kuvvetlendirici, bir otomobil, bir insan vücudu gibi), bir model ise bir “soyutlama”dır, matematiksel denklemlerin oluşturduğu bir kümedir.

Genellikle model, sistemin gerçek davranışına sadece yaklaşık bir davranış gösterir. Verilen bir sistem için, prensip olarak, her zaman bir model elde etmek mümkündür, ancak tersi doğru değildir, çünkü matematiksel denklemler her zaman gerçek sonuçlar vermez.

1.2 Sistemlerin Sınıflandırılması ve Zaman Kavramı

Sistemleri sınıflandırmanın yollarından biri zamandır.

Tanım 1.3 “Statik ve Dinamik Sistemler”:

Statik sistemlerde sistemin çıkışı, $y(t)$, geçmişteki giriş yada çıkış değerlerine bağlı değildir. Dinamik sistem de ise sistemin çıkışı, $y(t)$, geçmişteki giriş yada çıkış değerlerine bağlıdır.

Tanım 1.4 “Zamanla Değişen ve Değişmeyen Sistemler”:

Sistemin çıkışı zamana göre değişiyorsa, $y = g(u, t)$, zamanla değişen sistemdir. Sistemin çıkışı zamana göre değişmiyorsa, $y = g(u)$, zamanla değişmeyen (stasyoner) sistemdir.

1.3 Durum Kavramı ve Durum Uzayı Modeli

Sistemlerin modellenmesinin temelini durum kavramı oluşturur.

Tanım 1.5 “Durum”:

Bir sistemin t_0 anındaki durumu, tek başına $u(t)$ ($t \geq t_0$) bilgisinden, $y(t)$ ($t \geq t_0$) anındaki değerlerini belirlemeye yeten ve gereken bilgidir. Durum değişkenleri, $x(t) = [x_1, x_2, \dots, x_n]^T$ olarak tanımlanır.

Tanım 1.6 “Durum Denklemleri”:

Verilen $x(t_0)$ değerinden $t \geq t_0$ için $x(t)$ 'nin değerlerini belirlemek için gerekli olan denklem kümesine “durum denklemleri” denir.

Tanım 1.7 “Durum Uzayı”:

Bir sistemin “durum uzayı” genellikle X ile gösterilir, durumların mümkün olan bütün değerlerinin yer aldığı kümedir. Durum denklemleri,

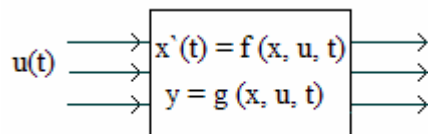
$$\dot{x}(t) = f(x(t), u(t), t) \quad (1.1)$$

durum uzayı modeli ise durum ve çıkış denklemleri ile tanımlanır.

$$\dot{x}(t) = f(x(t), u(t), t) \quad x(t_0) = x_0 \text{ (başlangıç koşulu)} \quad (1.2)$$

$$y(t) = g(x(t), u(t), t) \quad (1.3)$$

Verilen bir sistemin bir tek durum uzayı gösterilimi dolayısıyla modeli yoktur. Ancak genellikle doğal fiziksel büyüklüklerin durum değişkenleri olarak seçildiği modeller tercih edilir. Statik sistemlerde $\dot{x}(t) = 0$ olacaktır, yani $x(t) = st$ demektir ve sistem sadece çıkış denklemleri ile belirlenir. Zamanla değişmeyen bir sistemde ise f ve g , t 'ye bağımlı değildir, $\dot{x}(t) = f(x(t), u(t))$ ve $y(t) = g(x(t), u(t))$ olacaktır.



Şekil 1.1: Durum Uzayı Modeli

Tanım 1.8 “Lineer ve Lineer Olmayan Sistemler”:

Bir sistemin lineer olmasının gerek ve yeter koşulu hem $f(\cdot)$ hemde $g(\cdot)$ fonksiyonlarının lineer olmasıdır. Lineer durumda (1.2) ve (1.3) denklemleri,

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \quad (1.4)$$

$$y(t) = C(t)x(t) + D(t)u(t) \quad (1.5)$$

haline gelir. Lineer sistemler kümesi tüm sistemler içerisinde çok küçük bir kümeyi oluşturur, diğerleri lineer olmayan sistemler olarak sınıflandırılırlar.

Durum değişkenleri, reel sayılar veya reel değişkenli fonksiyonlar olabileceği gibi ayrık küme veya tamsayı kümesi de olabilirler, {AÇIK, KAPALI}, {YÜKSEK, ORTA, ALÇAK} veya {YEŞİL, KIRMIZI, MAVİ} gibi. Gerçekte unutulmamalıdır ki modelleme işlemi durumların, çıkışların veya girişlerin tanımlanması konusunda alabildiğine bir esnekliğe izin verirler. İlgilenilen program veya uygulamaya bağlı olarak uygun modelleme seçilir.

Sistemleri sınıflandırmanın diğer bir yolu ise model olarak seçilen problemin doğasına dayanır.

Tanım 1.9 “Sürekli Durum Sistemleri”:

Sürekli durum uzayı modelinde, durum uzayı X , n -boyutlu reel veya bazen karmaşık sayıların sürekli bir uzayından oluşur. Genellikle X sonlu boyutludur ancak sonsuz olduğu durumlar da vardır. Bu da differensiyel denklemlerin kullanılmasını gerektirir.

Tanım 1.10 “Ayrık Durum Sistemleri”:

Ayrık durum uzayı modelinde, durum uzayı ayrık bir kümedir. Durum geçiş mekanizması basit lojik ifadelerle dayanmaktadır. Buna rağmen durum denklemlerinin formal olarak ifade edilebilmesi için izlenen matematiksel yöntem oldukça karmaşıktır.

Diğer taraftan, sürekli durum uzayı modelleri kolaylıkla differansiyel denklemlerle ifade edilebilirler.

1.4 Geribesleme Kavramı

Geribesleme kavramı, sistem davranışından mümkün bilgileri kullanarak, sürekli olarak kontrol girişini sistemi istenen davranışa götürmek üzere ayarlar. Geribeslemenin temel özelliği beklenmedik bozucular altında sistemin istenen biçimde çalışmasını sağlıyor olmasıdır. Geribesleme kullanmanın avantajları; sistemin istenen davranışı beklenmedik bozuculara, modelde varsayılan parametrelerde oluşan hatalara karşı daha az duyarlıdır, $y(t)$ çıkışı istenen referans işareti $r(t)$ 'yi “otomatik” olarak takip eder. Diğer taraftan geribesleme kullanmanın getirdiği bazı dezavantajlar da vardır; çıkışı gözlemek, ölçmek ve kontrolör içinde değerlendirmek için sensörler ve diğer karmaşık cihazlar gerekir, tüm sistemin davranışını etkileyen geribesleme işareti enerji gerektirir, geribesleme gerçekte istenmeyen sistem davranışları gibi bazı problemler yaratır, yani bazı problemleri düzeltirken bazı yeni problemler yaratır.

$$u(t) = \gamma(r(t), x(t), t) \quad (1.6)$$

1.5 Ayırık Olay Sistemleri

Şimdiye kadar anlatılan sistemlerde zaman sürekli bir değişken olarak ele alındı. Bu şekilde modellenen sistemler de diferansiyel denklemler gibi matematiksel bir temele oturmaktadır. Sistemlerin giriş ve çıkış değişkenlerinin zamanın ayırık anlarında tanımlanmış olduğu varsayılınsın. Bunun sonucu olarak, ayırık zaman sistemleri elde edilir. Ayırık zaman sistemlerini incelemenin nedenleri olarak; dijital bilgisayarların çalışma prensibi, diferansiyel denklemlerin nümerik çözümleri, dijital kontrol teknikleri ve dijital kontrolörler, bazı sistemlerin doğaları gereği ayırık zamanlı olmaları; “ekonomik sistemler” gösterilebilir.

Ayrık olay sistemlerinin durum uzayı, $\{0,1,2, \dots\}$ gibi ayırık bir kümeden oluşur ve durum geçişleri sadece zamanın ayırık noktalarında gözlemlenir. Bu durum geçişlerine ise “olaylar” atanır.

Tanım 1.11 “Olay Kavramı”:

“Olay” sezgisel temeli kuvvetli basit ve ilkel bir kavramdır. Sadece anlık olarak oluştuğu, bir durum değerinden diğer bir durum değerine geçişe neden olduğu vurgulanabilir. Bir olay, belirli bir etki üzerinden tanımlanabilir, “birisi bir butona basar” ya da doğal olarak bir olay beklenmedik bir şekilde ortaya çıkar, örneğin “çok

karmaşık nedenlerden ötürü bir bilgisayar çöker” ya da çeşitli koşulların bir sonucu olarak bir olayla aniden karşılaşılır, örneğin “bir tanktaki sıvı seviyesi verilen değeri aniden aşar”.

“e” sembolü bir olayı göstermek için kullanılır. Bir sistem bir çok olaydan etkileniyorsa, “E” olaylar kümesi tanımlanır. “E” ayrık bir kümedir.

Örnek 1.1 Rasgele Yürüyüş :

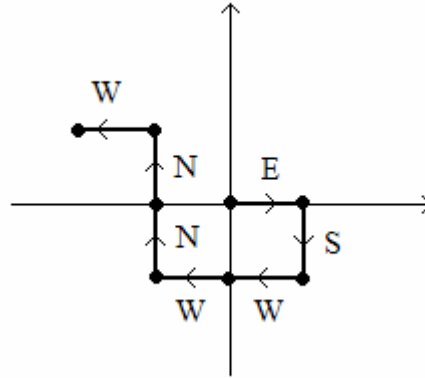
İki boyutlu bir alanda yapılan rasgele yürüyüş için, herhangi bir anda dört yöne (kuzey, güney, doğu, batı) doğru birim mesafe kadar hareket edilebildiği ve yönün rasgele seçildiği varsayımı yapılır. Sistemin durumu (x_1, x_2) konumlarıdır. x_1 ve x_2 sadece tam sayı değerleri alabilir. Yani durum uzayı ayrık bir kümedir.

$$X = \{ (i,j) : i,j = \dots, -1, 0, +1, \dots \} \quad (1.7)$$

Bu durumda, doğal bir olay kümesi;

$$E = \{N, S, W, E\} \quad (1.8)$$

olacaktır. Her bir olay herhangi bir yöne doğru yapılmış bir adım olarak tanımlanmıştır. (0,0) başlangıç durumunda sırasıyla {E, S, W, W, N, N, W} olayların olduğu varsayılırsa Şekil 1.2’deki durum yörüngesi (sample path) elde edilir.



Şekil 1.2: Rasgele Yürüyüş

1.5.1 Zaman Denetimli ve Olay Denetimli Sistemler

Sürekli durumlu sistemlerde durum genellikle zaman değiştiğinde, değişir. Bu aynı zamanda ayrık zaman modelleri için de doğrudur. Her saat tıklaması, durumda bir değişikliğe sebep olur. Çünkü “sürekli” durum değişkenleri zamana göre sürekli bir

şekilde değışirler. Bu nedenle bu sistemlere “zaman denetimli” sistemler denir. Bu durumda, zaman değışkeni ($t \in R$ veya $k \in I$ dir.) bağımsız değışken olarak ortaya çıkar ve giriş, durum ve çıkış fonksiyonlarının argümanını oluşturur.

Ayrık durum sistemlerinde durumlar sadece belirli zaman noktalarında ani geçişler şeklinde değışir. Bu ani her geçişe bir “olay” atanır. Bu geçişlerin “zamanlama mekanizması” iki durumda ele alınabilir;

- Her saat darbesinde, E’nin içinden bir olayın oluştuğu ve eğer hiçbir olay olmayacaksa, bir “boş olay” oluştuğu varsayılır. Boş olay E’nin bir üyesidir, özelliği ise hiç bir durum değışikliğine sebep olmamasıdır.
- Bir saat darbesiyle bağdaşmayan ve önceden bilinmesinin gerekli olmadığı birçok zamanda, bir olay “e”nin olduğu duyurulur.

Bu iki durum arasındaki temel farklar vardır.

İlk durumda, durum geçişleri bir saat ile senkronize edilmiştir. Her saat darbesinde seçilmiş bir olay oluşur ve sistem durum değıştirir ve bu işlem tekrarlanır. Saat, mümkün olan herhangi bir durum geçişinden tek başına sorumludur. İkinci durumda ise olaylar asenkron olarak ve birleşik olay süreçlerinin sonucu olarak meydana gelirler. Bu süreçlerinin birbirinden bağımsız olmasına gerek yoktur.

Bu iki durum arasındaki farklar zaman denetimli ve olay denetimli terimlere karşı düşen farklılıklardan kaynaklanır. Sürekli durumlu sistemler doğaları gereği “zaman denetimli” sistemlerdir. Oysa ayrık durum sistemlerinin hangi gruba dahil olduğu durum geçişlerinin bir saat darbesiyle mi yoksa asenkron olarak oluştuğuna bağlı olarak değışir. Bir saat darbesiyle oluşuyorsa zaman denetimli, asenkron olarak oluşuyorsa olay denetimlidir. Olay denetimli sistemlerin analizi ve modellenmesi çok daha karmaşıktır çünkü sistemin anlaşılabilmesi için belirlenmiş olan çok sayıda asenkron olay zamanlama mekanizması vardır.

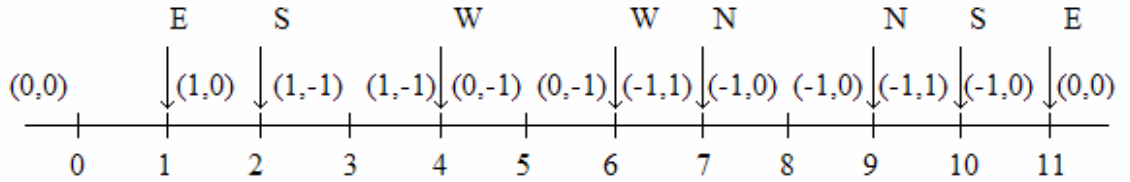
Olay denetimli sistemlerdeki durum geçişlerinin en iyi ve bilindik bir örneği bilgisayardaki “kesme” kavramıdır. Bir bilgisayarda bir çok işlem bir saat işareti ile senkronize edildiği, yani zaman denetimli olduğu halde, işletim sistemi zamanın herhangi bir anında oluşan çağrılara cevap verebilecek şekilde tasarlanmıştır. Bu çağrılar kullanıcı isteği ya da özel bir takım olayların sonucunda oluşabilir ancak bunlar bilgisayarın saat işaretinden tamamen bağımsızdır.

Örnek 1.2 Olay Denetimli Rasgele Yürüyüş:

Şekil 1.2’deki rasgele yürüyüş bir “zaman denetimli” sistemdir. Bir saat verilmiştir ve her saat darbesinde bir oyuncu bir parçayı hareket ettirir yani oyuncu E olay kümesi içinden bir olay seçer.

Ancak rasgele yürüyüşün alternatif bir biçimi daha vardır, burada parçacığın hareketinin kontrolü diğerinden farklıdır. Dört ayrı oyuncunun olduğu ve bunların herbirinin sadece bir yöne hareketten sorumlu olduğu, her oyuncunun tesadüfen davrandığı ve parçacığı kendi yönüne doğru hareket ettirdiği varsayılır. Bunun sonucu olarak, asenkron davranan oyuncular tarafından tanımlanan “olay denetimli” sistem ortaya çıkar.

N {7,9} ayrık zamanlarında, S {2,10}, W {4,6} ve E {1,11} de işaret verirse;



Şekil 1.3: Olay Denetimli Rasgele Yürüyüş

Örnek 1.2’de iki olayın tam olarak aynı anda oluşamayacağı varsayılmıştır. Eğer böyle bir durum olsaydı durum geçişi her iki olayın da olduğunu aksettirecek şekilde oluşmalıdır. Örneğin, 1 anında hem E hemde S’nin aynı anda oluştuğu varsayılırsa sonuç durum (1,-1) olacaktır. Ancak her zaman böyle olmaz, genel olarak iki farklı olayın durum üzerindeki etkisini bu olayların oluş sırası belirler. Mesela, durum bir banka hesabının dengesi olsun ve başlangıç olarak sıfır olduğu varsayılınsın. A olayı hesaba 100YTL para yatırılması, B olayı ise kredi kartı borcu olarak 100YTL çekilmesi olarak tanımlansın. Bu iki olayın aynı anda olduğu varsayılırsa, bu olayların oluş sıraları hesap dengesini etkiler. Eğer A olayı önce olursa net etki sıfır olur. Eğer B olayı önce olursa hesap önce eksiye düşeceği için faiz ödenecektir.

Bu gibi durumlarda iki olayın aynı anda olması ile farklı sıralarda olmasının etkilerinin ayrı ayrı modellenmesi gerekir.

1.5.2 Ayrık Olay Sistemlerinin Karakteristik Özellikleri

Sistem ve kontrol mühendisliğinin bugünkü başarısının altında yatan gerçek diferansiyel veya fark denklem temelli modellerdir. Bu matematiksel uygunluğa sahip modelleri kullanmak için,

- Sistem “sürekli durumlu” olmalıdır.

$x(.)$ ’lerin sürekli değişkenler olması demektir, $x(.) \in R$ veya $x(.) \in C$ olabilir. Bu nedenle bu türden sistemlere CVDS “Count Variable Dynamical System” denir. Fiziksel büyüklüklerin çoğu bu kategoriye girerler. Sürekli değişkenlerde türevlerinin alınabilir olması nedeniyle diferansiyel denklem kullanılabilir.

- Durum geçişlerinin mekanizması “zaman denetimli” olmalıdır.

Durumların değişmesi zamanla olur, t veya k zaman değişkeni bu sistemlerin modellenmesinde bağımsız değişkendir.

CVDS’nin aksine DEDS (Discrete Event Dynamical System) yada DES’de ise;

- Durum uzayı ayrıktır.
- Durum geçişleri “olay denetimli”dir.

Bu özelliklere dayanarak DES’in informal tanımı verilebilir.

Tanım 1.12 “DES”:

Bir DES; ayrık durumlu, olay denetimli sistemdir yani durum değişimi zaman boyunca asenkron olarak oluşan ayrık olaylara bağlıdır.

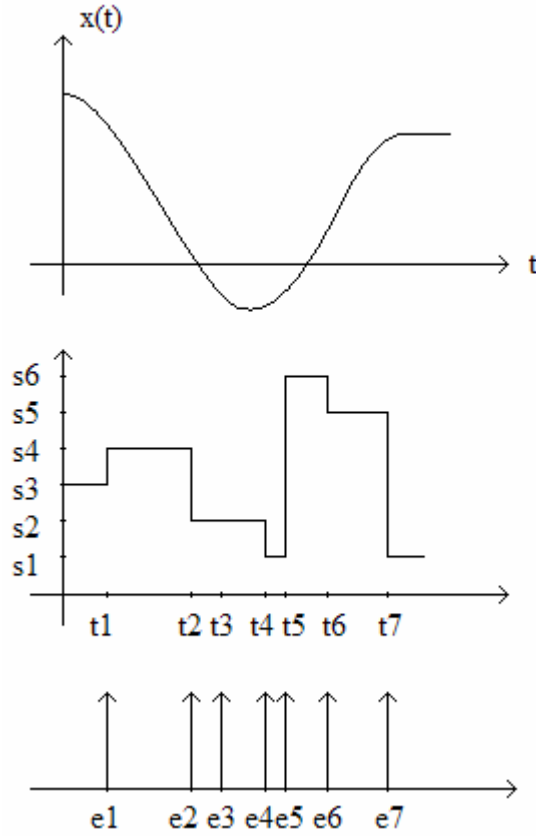
Bir çok sistem özellikle teknolojik olanlar, gerçekte ayrık durum sistemleridir. Doğaları gereği öyle olmasalar bile karmaşık bir sistemin bir ayrık durum görünümüyle ilgilenen bir çok uygulama vardır.

Bir makinanın durumu {On, Off} ya da {Meşgul, Boşta, Arızalı} olabilirken, bir programı çalıştıran bir bilgisayar şu üç durum içerisinde değerlendirilebilir {Giriş bekliyor, Çalışıyor, Bozuk}. Oyunların çoğu bir ayrık durum uzayına sahip olarak modellenebilir. Satrançta, örneğin her mümkün konfigürasyon bir durum olarak tanımlanır, oluşan durum uzayı çok geniştir ancak sonludur.

Olay denetimli sistemlerde durumlar zamanın sadece ayrık anlarında değişirler ve bu anlar “ayrık olayların” asenkron olarak oluşmalarına karşı düşen zaman noktalarıdır. Eğer bir durum geçişine neden olan her bir olay ile bir “olaylar” kümesi tanımlanabilirse, zaman bu sistemin denetiminde kullanılacak uygun bir bağımsız değişken olamaz.

CVDS ile DES'i ayıran iki temel özellik Şekil 1.5 de gösterildiği gibi her sistem sınıfına ilişkin tipik yörüngelerinin karşılaştırılması ile açığa çıkar.

- CVDS de durum uzayı $x \in \mathbb{R}$ reel sayılar kümesidir ve $x(t) \in X$ değerini alır. $x(t)$ fonksiyonu genel olarak $\dot{x}(t) = f(x(t), u(t), t)$ şeklindeki bir diferansiyel denklemin çözümüdür. $u(t)$ giriş fonksiyonudur.
- DES'de durum uzayı ayrıktır, $X = \{s_1, s_2, s_3, s_4, s_5, s_6\}$. Yörünge bir durumdan diğerine atlamalar şeklindedir ve bu atlama işlemi bir olay olduğunda gerçekleşir. Bir olay olduğu halde bir durum geçişi olmamış olabilir, e_3 de olduğu gibi. Bu noktada $\dot{x}(t) = f(x(t), u(t), t)$ gibi bir denkleme sahip olunamaz. Bu tür durumlarda olayların zaman boyunca nasıl davranacağını belirleyen bir mekanizma elde edilemez.



Şekil 1.4: DES ve CVDS Yörüngeleri

Bazı durumlarda sistem tümüyle CVDS olduğu halde sadece tesadüfen oluşan bazı ayrık olaylar nedeniyle başka bir yörüngeye sıçrama yapabilir. Bu olaylar bir çalışma modundan (durum denklemleri) başka bir çalışma moduna geçişi sağlarlar. Bu sistemlere “Hibrit Sistemler” denir.

Ayrıca ayrık zaman sistemler ile ayrık olay sistemlerinin karıştırılmaması gerekir. Ayrık zaman sistemleri hem CVDS ve hem de DES'i içerirler. DES ve CVDS hem sürekli hemde ayrık zamanlı olarak modellenenbilirler. Bir DES'i oluşturan ayrık olaylar eğer reel zaman anlarında oluşurlarsa, DES'in sürekli zaman modeli elde edilmiş olur.

1.5.3 Ayrık Olaylı Sistemlerde Üç Seviyede Soyutlama

Verilen bir sistemde yürütülebilecek tüm “zamanlanmış olaylar dizilerinin” yer aldığı kümeye sistemin “zamanlanmış dil” modeli denir. Olay kümesi ‘E’ ve bu olayların oluşturduğu sonlu olay dizisine ise “kelime” denir.

Sistemin yörüngeler kümesi hakkında eğer istatistiksel bilgiler varsa yani bir olayın oluşması olasılıklara bağlı ise bu sistemin modeline “stokastik zamanlanmış dil” denir. Bu model en ayrıntılı modeldir ve olay bilgilerini (olayların sırası ve oluşma biçimi), olayların zaman bilgilerini ve olayların başarılı oluşması hakkında istatistiksel bilgileri içerir.

Stokastik zamanlanmış dil modelinden istatistiksel bilgiler çıkartılırsa “zamanlanmış dil” modeline geçilir. “Zamanlanmış dil” modelinden de zaman bilgileri çıkartılırsa “zamanlanmamış dil” kısaca “dil” elde edilir.

“Diller”, “zamanlanmış diller” ve “stokastik zamanlanmış diller” DES'lerin modellenmesi ve üzerinde çalışabilmesi için kullanılan üç seviyeli soyutlamadır. Bu soyutlamalardan hangisinin seçileceği, analizin amacına bağlıdır. Her üçü seviyede birbirini tamamlayıcı niteliklere sahiptir.

1.6 Otomatlar ve Petri Ağları

Bu çalışmada, iki farklı “ayrık olay modellemesi” üzerinde durulacaktır: Otomatlar ve Petri Ağları. Bu biçimsel yapılar ortak bir gerçeğe sahiptir. Bu gerçek de dilleri bir durum geçiş yapısı kullanarak göstermesidir. Biçimsel yapılar gösterdikleri durum bilgileri ile birbirinden ayrılırlar. Bu yapılar ayrıca, sistem bileşenlerinin ayrık olay modellerinden hareketle bir sistemin ayrık olay modelinin inşaa edilmesine izin veren çeşitli operasyonların kompozisyonuna da uygundur. Bu özellik otomat ve petri ağlarını model kurma için uygun kılar. Analiz ve sentez konularına, modeldeki geçiş yapılarının yapısal özellikleri kullanılarak geçilebilir.

Bir DES'in tanımında iki özellik büyük önem taşır:

- Ayrık bir durum uzayı, X ile gösterilir.
- Ayrık bir olay kümesi, E ile gösterilir.

Ayrık Olay Sistemleri'nde (Discrete Event Systems (DES)) ilk hedef bu sistemlerin davranışlarını tanımlayan aynı zamanda tasarım, kontrol ve performans hedeflerini karşılayan modeller geliştirmektir.

Bir DES'in davranışı ' $e_1, e_2, \dots, e_n$ ' olay dizisi, "dil", ile belirlenir. Bu dil çeşitli olayların zaman içerisinde hangi sırayla oluştuğunu belirler ancak bu olayların oluştuğu zamana ilişkin bir bilgi içermez. Bir sistemin ne zaman bir duruma girdiği ya da ne kadar süre bu durumda kaldığı önemli değildir. Önemli olan durum değişikliklerine sebep olan olaylar dizisi ve bu olaylara karşı düşen yeni durumlarıdır.

Otomatlar ve Petri ağları DES'leri diller üzerinden modelleyen iki ana formalizmdir. Otomatlar analiz ve kontrol için uygun bir yapıya sahiptir, kullanımı kolaydır. Ancak yapısal eksikliklerinden dolayı durum uzayları oldukça büyüyebilir. Petri ağları ise daha fazla yapısal özellik taşımalarına rağmen otomatlar kadar analitik güce sahip değildir.

1.7 Tez Çalışmasının Amacı ve Elde Edilen Sonuçlar

Bu çalışmada, biçimsel işaretli bir dilden endüstriyel bir işlemciye otomatik kod üreten bir program geliştirme amaçlanmıştır. Bu amaç için, önce işaretli dilden üretilen dile geçilmiş ve bu dili üreten otomatın durum geçiş diyagramı elde edilmiştir. Daha sonra durum geçiş matrisinden sistemin Petri ağ modeli elde edilmiş ve bu model temel alınarak kullanılan işlemciye uygun kod üreten yazılım gerçekleştirilmiştir. Bu işlemler, tüm aşamaları görsel olarak sunmaya imkan veren Borland C++ Builder ortamında gerçekleştirilmiştir. Endüstriyel işlemci olarak SIMATIC-300 seçilmiş ve standart bir dil olan SCL dilinde kod üretilmiştir. Benzer yazılımlarda ortaya çıkan ve ilgili literatürde "çığ etkisi" (avalanche effect) olarak adlandırılan problem analiz edilmiş ve çığ etkisinden arındırılmış SCL kodu üretilmiştir.

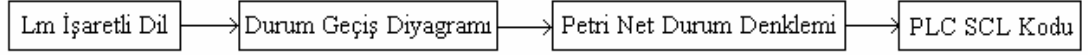
Bu çalışma sonucu elde edilen yazılımın oluşturulması sırasında izlenen yöntemle ilişkin işaret akış diyagramı Şekil 1.5'de gösterildiği gibidir. Öncelikle, girilen bir işaretli dilin otomatı, "G", belirlenir.

$$G = \{X, E, f, \Gamma, x_0, X_m\} \quad (1.9)$$

Daha sonra G otomatının durum geiş diyagramından “N” Petri ağına geilir. Durum denklemleri ($X_{k+1} = X_k + u_k A$) kullanılarak oluşan olaylara karşı düşen yeni durumlar hesaplanır.

$$N = (P, T, A, w, E, l, x_0, X_m) \quad (1.10)$$

Hem durum geiş diyagramı hem de Petri ağ durum denklemleri kullanılarak otomatın PLC’de çalışmasını sağlayacak program SCL dilinde üretilir.



Şekil 1.5: İşaret Akış Diyagramı

Oluşturulan yazılım çeşitli endüstriyel örnekler üzerinde denenmiş ve başarılı sonuçlar alınmıştır.

2. DİLLER VE OTOMATLAR

2.1 Sonlu Durum Sistemler ve Dil Modelleri

Her DES’e ilişkin bir “E”, olay kümesi mevcuttur. Bu kümenin sonlu olduğu varsayılır. Olay kümesi dilin “alfabe”si, olay dizileri ise dildeki “kelime”ler olarak adlandırılır. Herhangi bir olay içermeyen kelimeye “boş kelime” denir, “ε” ile ifade edilir. Bir kelimenin uzunluğu ise o kelime içindeki harf yani olay sayısına eşittir, aynı olayın tekrarları da bu sayıya dahil edilir ve $|\delta|$ ile ifade edilir. Boş kelimenin uzunluğu sıfırdır.

$$|\varepsilon| = 0 \quad (2.1)$$

Tanım 2.1 “Dil”:

E olay kümesi üzerinde tanımlıdır ve bu olay kümesine ait sonlu uzunluktaki kelimelerden oluşur, L ile ifade edilir.

$E = \{a, b, c\}$ olay kümesine ait birden fazla dil tanımlanabilir.

$$L_1 = \{\epsilon, a, acc\} \quad (2.2)$$

$$L_2 = \{a \text{ ile başlayan ve uzunluğu üç olan tüm kelimeler}\} \quad (2.3)$$

Kelime ve dolayısıyla dil oluşturmadaki en önemli işlem “türetme” işlemidir. L_1 dilindeki acc kelimesi ac kelimesi ve c olayından türetilmiştir, ac kelimesi ise a ve c olaylarından türetilmiştir.

Tanım 2.2 “Türetme”:

u, v ve bunlardan türetilen uv farklı anlamlara gelen üç ayrı kelimedir, uv türetmesi u ve v olaylarının arka arkaya gelmesiyle oluşur. Boş kelime, ϵ , türetme işleminin etkisiz elemanıdır.

$$u\epsilon = \epsilon u = u \quad (2.4)$$

E olay kümesi elemanlarının oluşturduğu tüm sonlu kelimeler kümesine boş kelime, ϵ , eklenirse E^* kümesi elde edilir. * işlemi “Kleene kapanışı” olarak adlandırılır. E^* keyfi uzunluğa sahip kelimelerden oluştuğu için sayılabilir sonsuz bir kümedir.

$E = \{a, b, c\}$ olay kümesi için E^* ,

$$E^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \dots\} \quad (2.5)$$

şeklinde tanımlanır.

E alfabesi ile oluşturulan bütün diller E^* ’ın bir alt kümesidir. Özel olarak $\emptyset$, E, E^* kümeleri de birer dildir.

t, u, v $\in E^*$ olmak üzere tuv = s yeni kelimesinde,

- t, ön ek
- u, alt kelime
- v, son ek olarak adlandırılır.

Her s ve ϵ , s kelimesinin ön ek, alt kelime ve son ekidir.

2.1.1 Diller Üzerindeki İşlemler

E^* kümesi üzerinde birleşme, kesişme, fark ve tümlleme işlemleri yapılabilir çünkü dil de bir kümedir. Bunlara ek olarak aşağıdaki işlemler de diller üzerinde tanımlıdır.

- Türetme: $L_a, L_b \subseteq E^*$ ise
 $L_a L_b := \{ s \in E^* : (s = s_a s_b) \text{ ve } (s_a \in L_a) \text{ ve } (s_b \in L_b) \}$ 'dir. $L_a L_b$ de tanımlı bir kelime, L_a ve L_b de tanımlı iki kelimenin türetilmesiyle oluşur.
- Ön Ek Kapanışı: $L \subseteq E^*$ ise $\bar{L} := \{ s \in E^* : \exists t \in E^* (st \in L) \}$ 'dir. L 'nin ön ek kapanışı, $\bar{L}$, L 'deki bütün kelimelerin ön eklerini içeren dildir. Genellikle $L \subseteq \bar{L}$ 'dir. $L = \bar{L}$ ise L 'ye "Ön Ek Kapalı" dil denir.
- Kleene Kapanışı: $L \subseteq E^*$ ise $L^* := \{ \epsilon \} \cup L \cup LL \cup LLL \cup \dots$ 'dir. E olay kümesinden E^* oluşturulması ile aynı işlemdir ancak E 'deki kelimelerin uzunluğu bir ile sınırlıdır. L^* 'ın elemanları, L elemanlarının sonlu sayıda türetilmesiyle oluşur. ** işlemi, $(L^*)^* = L^*$ özelliğine sahiptir .

Diller üzerinde gösterilen çeşitli işlemlerin ifadelerinde ön ek ve Kleen kapanışı önce uygulanmalı türetme işlemi birleşme, kesişme ve küme farkı gibi işlemler takip etmelidir.

Örnek 2.1:

$E = \{a, b, g\}$ ve bu E üzerinden tanımlanmış iki dil $L_1 = \{\epsilon, a, abb\}$ ve $L_2 = \{g\}$ olsun. $ab \notin L_1$ ve $\epsilon \notin L_2$ olduğu için L_1 ve L_2 ön ek kapalı değildir.

$$\begin{aligned} L_1 L_2 &= \{g, ag, abbg\} \\ \overline{L_1} &= \{\epsilon, a, ab, abb\} \\ \overline{L_2} &= \{\epsilon, g\} \\ L_1 \overline{L_2} &= \{\epsilon, a, abb, g, ag, abbg\} \\ L_2^* &= \{\epsilon, g, gg, ggg, \dots\} \\ L_1^* &= \{\epsilon, a, abb, aa, aabb, abba, abbabb, \dots\} \end{aligned}$$

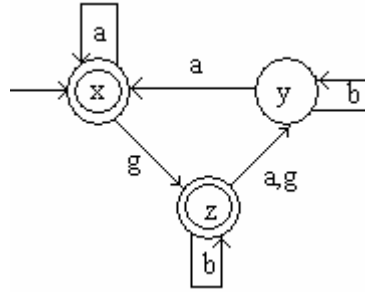
Boş dil ve boş kelimeye ilişkin tanımlamalar;

- $\epsilon \notin \emptyset$, boş kelime boş kümenin elemanı değildir.
- $\{\epsilon\}$ boş olmayan bir kelimedir ve sadece boş kelime içerir.

- iii. Eğer bir dil boş ise, $L=\emptyset$, ön ek kapanışı da boştur, $\bar{L}=\emptyset$. Eğer boş değilse, $L\neq\emptyset$, her zaman $\varepsilon \in \bar{L}$ olacaktır.
- iv. $\emptyset^*=\{\varepsilon\}$ 'dir, boş kümenin Kleen kapanışı tanım gereği ε 'u içerir. Yine tanım gereği $\{\varepsilon\}^* = \{\varepsilon\}$ ve $\varepsilon \cup \varepsilon = \varepsilon$ olacaktır.

2.2 Otomatlar

Bir otomat, iyi tanımlanmış kurallar üzerinden bir dili ifade etme kapasitesine sahip bir araçtır. Otomatları ifade etmenin en basit yolu durum geçiş diyagramlarını tanımlamaktır.



Şekil 2.1: Durum Geçiş Diyagramı

Şekil 2.1'deki yuvarlaklar durumları, oklar ise bu durumlar arasındaki geçişleri ifade eder. Bu şekil bir otomatın tam tanımını vermektedir. Bir otomatın durum kümesi yuvarlakların kümesine eşittir, $X=\{x, y, z\}$. Geçiş etiketleri otomatın alfabesini oluşturur, $E=\{a, b, g\}$. Oklar ise f geçiş fonksiyonlarını temsil eder.

$$f: X \times E \rightarrow X: f(x,a)=x, f(x,g)=z, f(y,a)=x, f(y,b)=y, f(z,b)=z, f(z,a)=f(z,g)=y \quad (2.6)$$

$f(y,a) = x$ geçiş fonksiyonu, otomat y durumunda iken bir a olayı oluştuğunda otomatın x durumuna geçeceğini belirtir. Bu a olayının nasıl oluştuğu önemsizdir.

Şekil 2.1 incelendiğinde, $f(x,a) = x$ gibi durum değişikliğine sebep olmadan da bir olayın meydana gelebildiği yada belirli bir durumda meydana gelen iki farklı olayın aynı durum değişikliğine sebep olabildiği görülebilir, $f(z,a) = f(z,g) = y$. Otomatın z 'den y 'ye geçişini gözlemleyerek a ve g olaylarından hangisinin meydana geldiği ayırt edilemez. Geçiş fonksiyonları $X \times E$ de tanımlanmış kısmi fonksiyonlardır bu yüzden E 'deki her olay için bir geçiş fonksiyonu tanımlamaya gerek yoktur. Şekil 2.1 de $f(x,b)$ ve $f(y,g)$ fonksiyonları tanımlanmamıştır.

Başlangıç durumu x_0 bir okla, işaretli durumlar X_m ise çift çemberle belirtilmiştir. Bir duruma özel bir anlam verilmek istenirse işaretli durum olarak tanımlanmalıdır. İşaretli durumlar “kabul gören” yada “son durumlar” olarak adlandırılırlar.

Tanım 2.3 “Belirgin Otomat”:

Belirgin otomat, “G” ile ifade edilir.

$$G = \{X, E, f, \Gamma, x_0, X_m\} \quad (2.7)$$

X : Durumlar kümesi,

E : Durum geçişlerine karşı düşen olaylar kümesi,

$f: X \times E \rightarrow X$ geçiş fonksiyonları,

$\Gamma: X \rightarrow 2^E$ aktif olay kümesi,

x_0 : Başlangıç durumu,

$X_m \subseteq X$ işaretli durumlar kümesidir.

- G , “durum makinesi” olarak da adlandırılır ve yukarıdaki gibi tanımlanır.
- Eğer X durum kümesi sonlu ise G ’ye “belirgin sonlu durum otomat” denir.
- Belirgin (deterministik) olmasının sebebi, f ’in $X \times E$ üzerinden tanımlanmış olmasıdır. Geçiş fonksiyonları $X \times E \rightarrow X$ yada $X \times E \rightarrow 2^X$ üzerinde tanımlanmış olan otomatlara “belirgin olmayan” otomat denir.
- f , geçiş fonksiyonlarının kısmen tanımlı olmasına izin verilir.
- Γ , aktif olay kümesi bir x durumunda oluşan ve durum değişikliğine sebep olmayan e olaylarını ile x durumunda aktif olmayan e' olaylarını ayırt etmeye yarar. $f(x, e) = x$ ’dir fakat $f(x, e')$ tanımsızdır.
- Uygun işaretli durumları, X_m , seçmek probleme bağlı olarak değişir. İşaretli durumlara girildiğinde sistem bazı görevlerin yada işlemlerin bittiğini belirleyebilir.

G otomatı başlangıç durumu x_0 ’dan başlar ve $e \in \Gamma(x_0) \subseteq E$ olayı gerçekleştiğinde $f(x_0, e) \in X$ durumuna geçer ve bu f ’in tanımlı olduğu tüm geçişler için devam eder.

$$f(x, e) := x,$$

$$s \in E^* \text{ ve } e \in E \text{ için } f(x, se) := f(f(x, s), e) \text{’dir.} \quad (2.8)$$

2.3 Otomatlarla Gösterilen Diller

Durum geçiş diyagramında başlangıç durumundan başlayarak izlenen tüm yollar “üretilen dil”i, işaretli durumlara giden tüm yollar ise “işaretlenen dil”i ifade eder.

Tanım 2.4 “Üretilen ve İşaretlenen Diller”:

$G = \{X, E, f, \Gamma, x_0, X_m\}$ tarafından üretilen dil $\mathcal{L}(G)$, işaretlenen dil ise $\mathcal{L}_m(G)$, şeklinde ifade edilir. $f: X \times E \rightarrow X$ olarak alınmıştır.

$$\mathcal{L}(G) := \{s \in E^* : f(x_0, s) \text{ tanımlı} \} \quad (2.9)$$

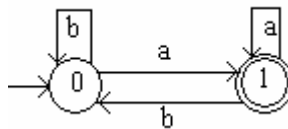
$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\} \quad (2.10)$$

Bir yola ilişkin kelime, o yolu oluşturan geçiş etiketlerinin yani olayların birleşiminden oluşur. Bu durumda bir s kelimesi yalnızca durum geçiş diyagramında erişilebilir bir yola karşılık düşüyorsa $\mathcal{L}(G)$ içinde tanımlıdır. $\mathcal{L}(G)$, tanım gereği ön ek kapalıdır çünkü bir yolun mümkün olması ancak ön eklerinin de mümkün olmasıyla sağlanabilir. Eğer f kısmi değil de tüm bir fonksiyon olarak tanımlanırsa $\mathcal{L}(G) = E^*$ olur.

$\mathcal{L}_m(G)$, $\mathcal{L}(G)$ ’nin alt kümesidir, $f(x_0, s) \in X_m$ şartını sağlayan yani durum geçiş diyagramında işaretli durumlara giden tüm kelimeleri içerir. Tüm durumların işaretli olması gerekmediği için $\mathcal{L}_m(G)$ ’nin de ön ek kapalı olması gerekli değildir. İşaretlenen dil otomat tarafından “tanınan dil” yada otomatın “tanıdığı” olarak adlandırılır.

Örnek 2.2: İşaretlenen Dil

$E = \{a, b\}$ olay kümesi üzerinde tanımlanmış $L = \{a, aa, ba, aaa, aba, baa, bba, \dots\}$ dili, a ve b olaylarından oluşan ama her zaman a olayı ile biten kelimeleri içerir. Bu dili işaretleyen bir $G = \{X, E, f, \Gamma, x_0, X_m\}$ otomatı Şekil 2.2’deki gibi sonlu durumlu olabilir.

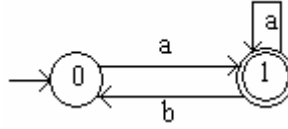


Şekil 2.2: Örnek 2.2’nin Durum Geçiş Diyagramı

Bu otomatta, $X=\{0,1\}$, $x_0=0$, $X_m = \{1\}$ ve $f(0,a)=1$, $f(0,b)=0$, $f(1,a)=1$, $f(1,b)=0$ olarak tanımlıdır. Durum geçiş diyagramından $\mathcal{L}_m(G) = L$ olduğu görülür. Geçiş fonksiyonları, f , tüm olduğu için G tarafından üretilen dil $\mathcal{L}(G) = E^*$ 'dir.

Örnek 2.3: Üretilen ve İşaretlenen Dil

Şekil 2.2'de 0 durumundaki b geçişi kaldırılırsa Şekil 2.3'deki durum geçiş diyagramını elde edilir.



Şekil 2.3: Örnek 2.3'nin Durum Geçiş Diyagramı

$\mathcal{L}(G)$, E^* 'da a ile başlayan ve b olayının arka arkaya oluşumunu içermeyen kelimelerden oluşur. $f(0,b)$ geçişi tanımsızdır ve $\mathcal{L}_m(G) \subseteq \mathcal{L}(G)$ 'dir.

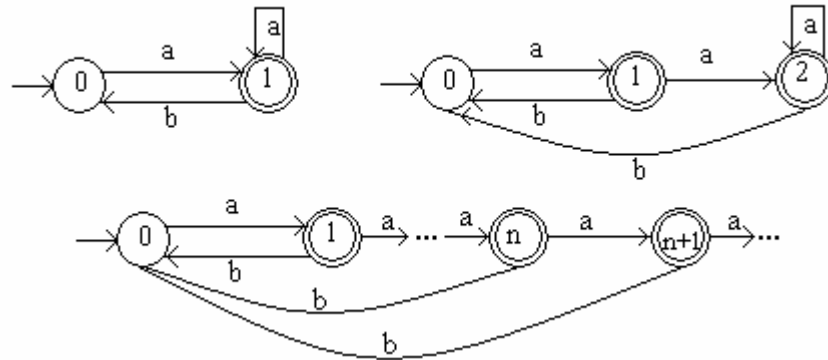
2.3.1 Otomatların Eşdeğerliği

Verilen dili üreten ve işaretleyen bir otomat oluşturma birinden fazla yolu olabilir. Eğer iki otomat aynı dili üretiyor ve işaretliyorsa birbirinin eşdeğeridir.

$$\mathcal{L}(G_1) = \mathcal{L}(G_2) \text{ ve } \mathcal{L}_m(G_1) = \mathcal{L}_m(G_2) \quad (2.11)$$

Örnek 2.4: Eşdeğer Otomatlar

Şekil 2.4'de durum geçiş diyagramları gösterilen tüm otomatlar birbirinin eşdeğeridir, çünkü hepsi aynı dili üretir ve işaretler.



Şekil 2.4: Üç Eşdeğer Otomat

2.3.2 Kilitlenme

Genellikle G , $\mathcal{L}(G)$ ve $\mathcal{L}_m(G)$ tanımları arasında $\mathcal{L}_m(G) \subseteq \bar{\mathcal{L}}_m(G) \subseteq \mathcal{L}(G)$ olduğu kabul edilir. İlk kısım tüm durumların işaretli olmadığı, ikinci kısım ise $\mathcal{L}_m(G)$ 'nin tanımını ve $\mathcal{L}(G)$ 'nin tanım gereği ön ek kapalı olmasından kaynaklanır.

Tanım 2.5 “Açmaz”:

Bir G otomatı, $\Gamma(x) = 0$ ve $x \notin X_m$ olan bir x durumuna erişilebiliyorsa buna “açmaz” denir, sistem kilitlenir ve hangi olay olursa oluştun durum değıştirmemez. Bu durumda $\bar{\mathcal{L}}_m(G) \subseteq \mathcal{L}(G)$ olacaktır. Çünkü x durumunda sonlanan herhangi bir $\mathcal{L}(G)$ kelimesi, $\mathcal{L}_m(G)$ 'deki herhangi bir kelimenin ön eki olamaz.

Tanım 2.6 “Kısır Döngü”:

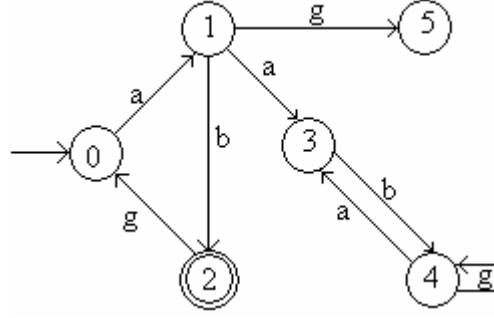
Bir G otomatında işaretli olmayan bazı durumlar kendi aralarında birbirlerine erişebiliyor ama başka bir duruma çıkış vermiyorlarsa buna “kısır döngü” denir. Otomat canlıdır, durum değıştirmeye devam eder ancak kilitlenmiştir çünkü işaretli durumlara erişemez. Bu durumda da $\bar{\mathcal{L}}_m(G) \subseteq \mathcal{L}(G)$ olacaktır. Çünkü kısır döngü oluşturan durumlarda sonlanan herhangi bir $\mathcal{L}(G)$ kelimesi, $\mathcal{L}_m(G)$ 'deki herhangi bir kelimenin ön eki olamaz.

Tanım 2.7 “Kilitlenme”:

Bir G otomatı $\bar{\mathcal{L}}_m(G) \subseteq \mathcal{L}(G)$ ise kilitlenebilir, $\bar{\mathcal{L}}_m(G) = \mathcal{L}(G)$ ise kilitlenemezdir. Bir otomatın kilitlenebilir olması açmaza yada kısır döngüye girebileceğı anlamına gelir.

Örnek 2.5: Kilitlenme

Şekil 2.5’de tanımlanmış G otomatında 5. durumda açmaz oluşur, sistem işaretli olmayan bu durumda kalır. 3.ve 4. durumlar ve bunlara ilişkin a, b, g geçişleri ise kısır döngüyü oluşturur. 3 ve 4 işaretli durumlar değildir ve sistem 3. duruma ulaştığında kısır döngü oluşur. $ag \in \mathcal{L}(G)$ fakat $ag \notin \mathcal{L}_m(G)$ olduğu için $\bar{\mathcal{L}}_m(G) \subseteq \mathcal{L}(G)$ ’dir.



Şekil 2.5: Açmaz ve Kısır Döngü Örnekleri

2.3.3 Belirgin Olmayan Otomatlar

Otomat tanımında, durum geçişleri herhangi bir x durumunda meydana gelen bir e olayının sistemi hangi yeni duruma getireceğini belirler. Ancak x durumunda oluşan e olayı sistemi birden fazla yeni duruma da götürebilir. Buna otomatlarda izin verilmemesinin sebebi göz ardı edilmek istenmesinden kaynaklanır. Bu durumda $f(x,e)$ artık yeni bir $x \in X$ durumunu değil, olası yeni durumlar $\{x\}$ kümesini tanımlar.

Aynı zamanda durum geçiş diyagramındaki bazı geçişlere ε , boş kelime, etiketi verilmek istenebilir. Bunlara “ ε -geçişleri” adı verilir. Bu geçişler sistemin iç durumunda değişikliğe sebep olsalar da sistem dışından gözlemlenemezler.

Tanım 2.8 “Belirgin Olmayan Otomat”:

Belirgin olmayan otomat, “ G_{nd} ” ile ifade edilir.

$$G_{nd} = \{X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_m\} \quad (2.12)$$

X : Durumlar kümesi,

$E \cup \{\varepsilon\}$: Durum geçişlerine karşı düşen olaylar kümesi,

$f_{nd}: X \times E \cup \{\varepsilon\} \rightarrow 2^X$, , $f_{nd}(x,e) \subseteq X$, geçiş fonksiyonları,

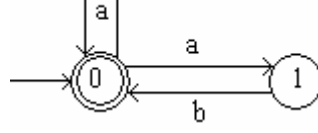
$\Gamma: X \rightarrow 2^E$, aktif olay kümesi,

x_0 : Başlangıç durumu, $x_0 \in X$,

$X_m \subseteq X$ işaretli durumlar kümesi’dir.

Örnek 2.6:

Şekil 2.6’da tanımlanmış sonlu durum otomatta, 0 durumunda oluşan a olayı sistemi hem 1’e götürebilir hem de 0’da bırakabilir. Geçiş fonksiyonları $f_{nd}(0,a) = \{0,1\} \subseteq X$ ve $f_{nd}(1,b) = \{0\} \in X$ olarak tanımlanır. $f_{nd}(0,b)$ ve $f_{nd}(1,a)$ tanımsızdır.



Şekil 2.6: Belirgin Olmayan Otomat Örneği

f_{nd} , u kelimesine göre genişletilirse ve $u = \varepsilon$ durumu da dahil edilirse;

$$f_{nd}(x, ue) := \{ z : z \in f_{nd}(y, e) \text{ bazı } y \in f_{nd}(x, u) \text{ için } \} \quad (2.13)$$

Öncelikle x durumundan u olayı ile ulaşılabilen tüm y durumları bulunur, daha sonra bu y durumlarından e olayı ile ulaşılabilen tüm z durumları bulunur. Şekil 2.6’daki örneğe göre $f_{nd}(0,ab)$ hesaplanmak istenirse; $f_{nd}(0,a) = \{0,1\}$, $f_{nd}(1,b) = \{0\}$ ve $f_{nd}(0,b)$ tanımsız olarak bulunur. Bu durumda $f_{nd}(0,ab) = \{0\}$ olur.

Belirgin olmayan otomatların ürettiği ve işaretlediği dil;

$$\mathcal{L}(G_{nd}) := \{s \in E^* : \exists x \in x_0 (f_{nd}(x, s) \text{ tanımlıdır})\} \quad (2.14)$$

$$\mathcal{L}_m(G_{nd}) := \{s \in \mathcal{L}(G_{nd}) : \exists x \in x_0 (f_{nd}(x, s) \cap X_m \neq \emptyset)\} \quad (2.15)$$

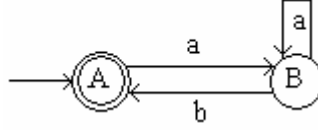
şeklinde tanımlanır. x_0 , belirgin olmayan otomatlar için küme olacaktır.

$\mathcal{L}(G_{nd})$, x_0 ’dan başlayan ve durum geçiş diyagramında bulunan geçişlere ilişkin kelimelerden; $\mathcal{L}_m(G_{nd})$ ise x_0 ’dan işaretli durumlara giden yollara ilişkin kelimelerden oluşur.

Örnek 2.7:

Şekil 2.7’de tanımlanmış belirgin otomat, G , Şekil 2.6’da tanımlanmış olan belirgin olmayan otomatın, G_{nd} , eşleniğidir. Her iki otomatta aynı dili üretir ve işaretler. G otomatındaki A durumu G_{nd} ’deki 0 durumuna, G ’deki B durumu

G_{nd} 'deki $\{0,1\}$ durumlarına karşılık düşer. Her iki otomatın geçiş fonksiyonları da birbirine karşılık düşer,



Şekil 2.7: Belirgin Otomat Örneği

- i. $f(A,a) = B$ ve $f_{nd}(0,a) = \{0,1\}$.
- ii. $f(A,b)$ ve $f_{nd}(0,b)$ tanımsızdır.
- iii. $f(B,a) = B$ ve $f_{nd}(0,a) = \{0,1\}$, $f_{nd}(1,a)$ tanımsızdır.
- iv. $f(B,b) = A$ ve $f_{nd}(1,b) = \{0\}$, $f_{nd}(0,b)$ tanımsızdır.

3. OTOMATLAR ÜZERİNDE İŞLEMLER

Otomatlar ile modellenen DES'lerin analizi için bir takım işlemlere gerek duyulur. Bunlar,

- Tek bir otomat üzerine işlemler
- İki veya daha çok otomat üzerine tanımlı işlemler

olarak iki ana başlık altında toplanabilir. Bu işlemleri kullanarak belirgin olmayan bir otomatın belirgin olan bir eşleniği bulunabilir.

3.1 Tekli İşlemler

Tekli işlemler bir otomatın durum geçiş diyagramını uygun şekilde değiştirmek için kullanılabilir.

Tanım 3.1 “Erişilebilir Kısım” :

$\mathbb{f}(G)$ ve $\mathbb{f}_m(G)$ tanımlarından görüleceği üzere, G tarafından işaretlenen ve üretilen dili etkilemeden, x_0 ’dan erişilemeyen tüm durumlar G ’den silinebilir ve böylece $Ac(G)$ elde edilir. Bir durum silindiğinde o duruma ilişkin tüm geçişler de silinmiş olur.

$$Ac(G) := (X_{ac}, E, f_{ac}, x_0, X_{ac,m}) \quad (3.1)$$

$$X_{ac} = \{x \in X : \exists s \in E^* (f(x_0, s) = x)\}$$

$$X_{ac,m} = X_m \cap X_{ac}$$

$$f_{ac} = f|_{X_{ac} \times E \rightarrow X_{ac}}$$

$\mathbb{f}(G)$ ve $\mathbb{f}_m(G)$, $Ac(G)$ işleminden etkilenmezler çünkü otomatı silinen durumlara götüren kelimeleri içermezler. Bu yüzden tüm otomatların “erişilebilir” olduğu kabul edilir, $G = Ac(G)$.

Tanım 3.2 “Tamamlayıcı Erişilebilir Kısım” :

G ’nin bir x durumu, eğer x ’den geçen bir $\mathbb{f}_m(G)$ kelimesi varsa, “tamamlayıcı erişilebilir”dir (coaccessible). Durum geçiş diyagramında $x \in X$ durumundan işaretli bir duruma, $x_m \in X_m$, gidilebileceği anlamına gelir. İşlem olarak ise, G otomatında tamamlayıcı erişilebilir olmayan durumları silme işlemine $CoAc(G)$, G ’nin $CoAc$ kısmını elde etme işlemi denir.

$$CoAc(G) := (X_{coac}, E, f_{coac}, x_{0,coac}, X_m) \quad (3.2)$$

$$X_{coac} = \{x \in X : \exists s \in E^* (f(x, s) \in X_m)\}$$

$$x_{0,coac} = \{x_0 \text{ eğer } x_0 \in X_{coac}, \text{ değilse tanımsız}\}$$

$$f_{coac} = f|_{X_{coac} \times E \rightarrow X_{coac}}$$

$CoAc$ işlemi sonucunda x_0 ’dan erişilebilen durumlar silinebileceği için, $CoAc$ $\mathbb{f}(G)$ ’yi etkiler ancak $\mathbb{f}_m(G)$ ’yi etkilemez çünkü çıkarılan durum x_0 ’dan $x_m \in X_m$ ’e giden herhangi bir yol üzerinde bulunamaz. Eğer $G = CoAc(G)$ ise G “tamamlayıcı erişilebilir”dir, bu durumda $\mathbb{f}(G) = \bar{\mathbb{f}}_m(G)$ olacaktır. Tamamlayıcı erişilebilirlik ve kilitlenme tanımları birbirleriyle yakından ilişkilidir. Bir otomat $\bar{\mathbb{f}}_m(G) \neq \mathbb{f}(G)$ ise kilitlenebilirdir yani tamamlayıcı erişilebilir değildir, eğer tamamlayıcı erişilebilir ise kilitlenemez.

CoAc(G) işlemi sonucunda $X_{coac} = \emptyset$ olursa bu durumda $X_m = \emptyset$ olacak ve “boş otomat” elde edilecektir. Boş otomat durum uzayı boş olan bir otomattır ve ürettiği ve işaretlediği küme boş küme olacaktır.

Tanım 3.3 “Budama İşlemi” :

Bir otomat, hem tamamlayıcı erişilebilir, $CoAc(G) = G$, hem de erişilebilir, $Ac(G) = G$, ise “budanmış” olarak adlandırılır. $G = Trim(G)$ ise G’ye “budanmış otomat” denir.

$$Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)] \quad (3.3)$$

Tanım 3.4 “Tümleme İşlemi” :

$L \subseteq E^*$ dilini işaretleyen bir $G = \{X, E, f, \Gamma, x_0, X_m\}$ budanmış otomatı varsa $E^* \setminus L$ dilini işaretleyen otomat, G^{comp} olarak adlandırılır. G^{comp} otomatı, E olay kümesi üzerinden tanımlanmış E^* kelime uzayından, $\mathbb{F}_m(G) = L$ ’de olanlar çıkartılarak $\mathbb{F}_m(G^{comp}) = E^* \setminus L$ olacak şekilde, 2 adımda inşa edilir.

- G’deki f geçiş fonksiyonlarını tümlenir ve tam bir fonksiyon olan f_{tot} elde edilir. Bu işlem için X durum uzayına x_d “ölü” durumu eklenir ve G’de tanımlanmamış tüm $f(x,e)$ ’ler x_d ’ye yönlendirilir.

$$f_{tot} = \{ f(x,e) \text{ eğer } e \in \Gamma(x), \text{ değil ise } x_d \} \quad (3.4)$$

Ayrıca, her $e \in E$ için $f_{tot}(x,e) = x_d$ olarak tanımlanır, x_d işaretli bir durum olarak tanımlanmaz. Bu şekilde $\mathbb{F}(G_{tot}) = E^*$ ve $\mathbb{F}_m(G_{tot}) = L$ olan, G_{tot} otomatı elde edilir.

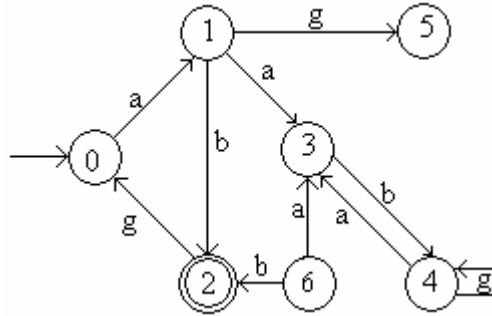
$$G_{tot} = (X \cup \{x_d\}, E, f_{tot}, x_0, X_m) \quad (3.5)$$

- G_{tot} otomatındaki tüm işaretli durumlar işaretsiz, tüm işaretsiz durumlar (x_d de dahil olmak üzere) işaretli durum olarak değiştirilir. Böylece $\mathbb{F}(G^{comp}) = E^*$ ve $\mathbb{F}_m(G^{comp}) = E^* \setminus L$ olacak şekilde G^{comp} otomatı tanımlanmış olur.

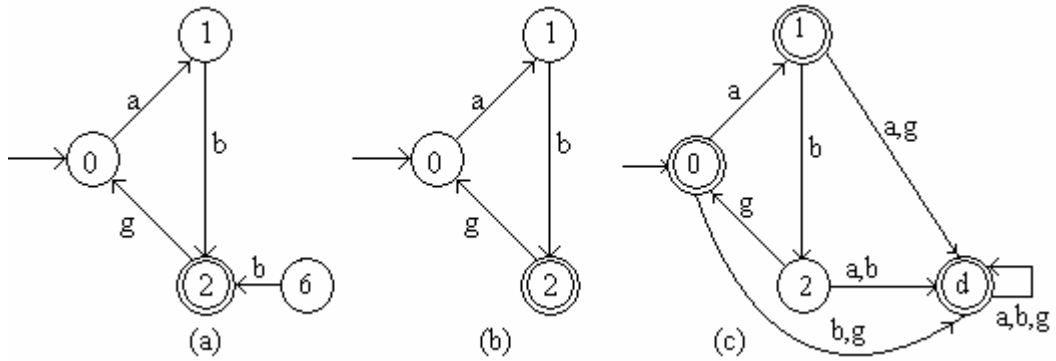
$$G^{comp} = (X \cup \{x_d\}, E, f_{tot}, x_0, (X \cup \{x_d\}) \setminus X_m) \quad (3.6)$$

Örnek 3.1:

Şekil 3.1’de tanımlanmış olan G otomatının $\text{CoAc}(G)$ kısmını (Şekil 3.2(a)) elde etmek için, işaretli durum olan 2’ye gitmeyen tüm durumlar yani 3, 4, 5 ve bu durumların geçiş fonksiyonları silinir. $\text{CoAc}(G)$ ’ye $\text{Ac}(G)$ ’yi uygulanır, başlangıç durumu 0’dan erişilemeyen durumlar yani 6. durum silinirse, Şekil 3.2(b)’deki $\text{Trim}(G)$ elde edilir. Şekil 3.2(c)’deki $\text{Trim}(G)$ ’nin tümleyenini elde etmek için ise, otomata yeni bir “d” durumu eklenir ve tanımsız olan tüm geçişler d’ye yönlendirilir.



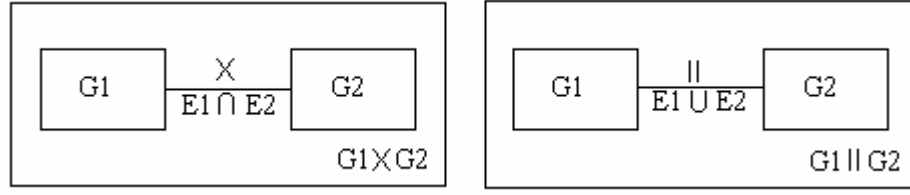
Şekil 3.1: G Otomatı



Şekil 3.2: (a) $\text{CoAc}(G)$, (b) $\text{Trim}(G)$, (c) $\text{Trim}(G)$ ’nin Tümleyeni

3.2 Kompozisyon İşlemleri

Birden fazla otomat üzerinde tanımlanabilen iki işlem vardır, çarpma “ $\times$ ” ve paralel kompozisyon “ $\parallel$ ”. Paralel kompozisyon “senkron kompozisyon”, çarpma ise “tümüyle senkron kompozisyon” olarak da adlandırılır. Bu işlemler, eş zamanlı çalışan otomatlar kümesinin ortak davranışını, $G_1 \times G_2$ ve $G_1 \parallel G_2$ olmak üzere iki ayrı biçimde modeller. $\times$ ile bağlanmış sistem $E_1 \cap E_2$ olay kümesi, $\parallel$ ile bağlanmış sistem $E_1 \cup E_2$ olay kümesi ile tanımlanır.



Şekil 3.3: İki Otomatın Bağlantısı

$G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ ve $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$ otomatlarının erişilebilir olduğu, $Ac(G_1) = G_1$ ve $Ac(G_2) = G_2$, kabul edilir ancak tamamlayıcı erişilebilir olmalarına gerek yoktur.

Tanım 3.5 “Çarpma İşlemi” :

G_1 ve G_2 otomatlarının çarpımı,

$$G_1 \times G_2 := Ac(X_1 \times X_2, E_1 \cap E_2, f, \Gamma_{1 \times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \quad (3.7)$$

$f((x_1, x_2), e) := \{(f_1(x_1, e), f_2(x_2, e)) \text{ eğer } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \text{ değilse tanımsız} \}$

ve $\Gamma_{1 \times 2}, (x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$ olarak tanımlanır. $X_1 \times X_2$ durum uzayı, X_1 ve X_2 durum uzayı elemanlarının (x_1, x_2) şeklinde çiftler halinde tanımlanmasıyla oluşur. G_1 otomati x_1 durumunda ve G_2 otomati x_2 durumunda iken bir e olayı oluştuğunda, eğer e olayı hem G_1 hem de G_2 otomatında (x_1, x_2) durumları için tanımlı ise otomatlar ait oldukları çifte giderler. Eğer e olayı otomatlardan birinde bile tanımlı değil ise $f((x_1, x_2), e)$ geçişi tanımsızdır.

Çarpma işleminde, iki otomatın geçişleri ortak olaylar kümesi $(E_1 \cap E_2)$ üzerinden daima senkronize olmalıdır. $G_1 \times G_2$ çarpımı, G_1 ve G_2 otomatlarının “uygun adım” bağlantısını temsil eder, bir olayın oluşması için o olayın her iki otomatta da oluşması gerekir.

İki dilin arakesiti ile oluşturulan bir dil, iki dilin otomatlarının çarpımı ile oluşturulur.

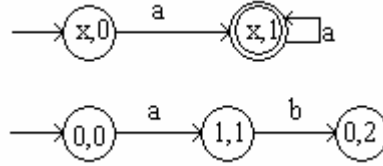
$$\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2) \quad (3.8)$$

$$\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2) \quad (3.9)$$

Eğer $E_1 \cap E_2 = \emptyset$ ise $\mathcal{L}(G_1 \times G_2) = \{\epsilon\}$; $\mathcal{L}_m(G_1 \times G_2) = \{\epsilon\}$ eğer (x_{01}, x_{02}) işaretli ise, değilse $\mathcal{L}_m(G_1 \times G_2) = \emptyset$ olarak tanımlanır.

Örnek 3.2:

Şekil 3.4'deki ilk otomat Şekil 2.1 ve Şekil 2.2 de tanımlanan otomatların çarpımı ile oluşturulmuştur. Ortak olay kümesi $E = E_1 \cap E_2 = \{a, b\}$ 'dir. $(x,0)$ başlangıç durumunda her iki otomatda da tanımlı olan tek olay, a olayıdır. Birinci otomatı x durumuna, 2. otomatı 1 durumuna getirir, dolayısıyla yeni durum $(x,1)$ olur. $(x,1)$ durumunda da tanımlı olan tek olay, a olayıdır. Birinci otomatı x durumuna, 2. otomatı 1 durumuna getirir, dolayısıyla yeni durum yine $(x,1)$ olur. $(x,1)$ durumu işaretli bir durumdur çünkü hem x hem de 1 kendi otomatlarında işaretli durumlardır. Şekil 3.4'deki ikinci otomat Şekil 2.2 ve Şekil 3.2(b)'de tanımlanan otomatların çarpımı ile oluşturulmuştur. Ortak olay kümesi $E = E_1 \cap E_2 = \{a, b\}$ 'dir. Uygun adım olan tek davranışları ab kelimesidir, bu da otomatı $(0,2)$ açmaz durumuna götürür. Şekil 3.4'deki ilk otomat Şekil 2.1 ve Şekil 2.2 de, ikinci otomat ise Şekil 2.2 ve Şekil 3.2(b) de tanımlanan otomatların çarpımıdır.



Şekil 3.4: Otomatların Çarpımı

Tanım 3.6 “Paralel Kompozisyon”:

G_1 ve G_2 otomatlarının paralel kompozisyonu,

$$G_1 \parallel G_2 := Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_1 \parallel_2, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \quad (3.10)$$

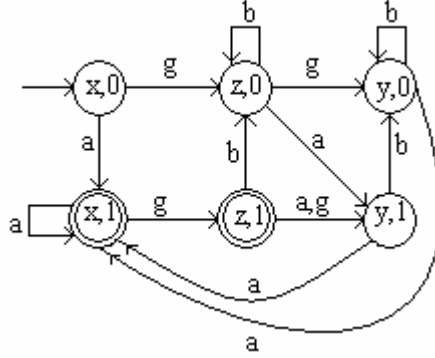
$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \text{eğer } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1, e), x_2) & \text{eğer } e \in \Gamma_1(x_1) \setminus E_2, \\ (x_1, f_2(x_2, e)) & \text{eğer } e \in \Gamma_2(x_2) \setminus E_1, \\ \text{tanımsız} & \text{diğer} \end{cases}$$

ve $\Gamma_1 \parallel_2(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$ olarak tanımlanır.

Paralel kompozisyon işleminde, iki otomatın geçişleri ortak olaylar kümesi $(E_1 \cap E_2)$ üzerinden daima senkronize olmalıdır. Ancak otomatlardan biri $(E_1 \setminus E_2) \cup (E_2 \setminus E_1)$ kümesindeki özel olaylarda diğer otomatın katılımı olmaksızın kendi özel olayını yürütebilir.

Eğer $E_1 = E_2$ ise paralel kompozisyon, çarpma işlemine indirgenir, çünkü bütün geçişler senkron olmaya zorlanır. Eğer $E_1 \cap E_2 = \emptyset$ ise senkronize geçişler yoktur, paralel kompozisyon G_1 ve G_2 otomatlarının “kıırma”sı olarak adlandırılır. Paralel kompozisyon yer değiştirme özelliğine sahiptir.

$$G_1 \parallel G_2 = G_2 \parallel G_1 \text{ ve } G_1 \parallel (G_2 \parallel G_3) = (G_1 \parallel G_2) \parallel G_3 \quad (3.11)$$



Şekil 3.5: Şekil 2.1 ve Şekil 2.2'deki Otomatların Paralel Kompozisyonu.

$G_1 \parallel G_2$ 'nin işaretlediği ve ürettiği dili $\mathcal{L}_m(G_1)$ ve $\mathcal{L}_m(G_2)$ cinsinden tanımlamak için önce “izdüşüm” ve “ters izdüşüm” tanımları yapılmalıdır.

Tanım 3.7 “İzdüşüm”:

İzdüşüm P_i ,

$$P_i : (E_1 \cup E_2)^* \rightarrow E_i^* \quad i=1,2$$

$$P_i(\varepsilon) := \varepsilon$$

$$P_i(e) := \{ e \text{ eğer } e \in E_i, \varepsilon \text{ eğer } e \notin E_i \} \text{ olarak tanımlanır.}$$

$$P_i(se) := P_i(s) P_i(e) \quad \text{her } s \in (E_1 \cup E_2)^*, e \in (E_1 \cup E_2) \text{ için doğrulanır.}$$

Tanım 3.8 “Ters İzdüşüm”:

Ters izdüşümü P_i^{-1} ,

$$P_i^{-1}: E_i^* \rightarrow 2^{(E_1 \cup E_2)^*}, P_i^{-1}(t) := \{ s \in (E_1 \cup E_2)^* : P_i(s) = t \} \text{ olarak tanımlanır.}$$

P_i ve P_i^{-1} işlemleri dillere genişletilebilir. L 'deki tüm kelimeler için P_i ve P_i^{-1} bulunursa dilin izdüşümleri bulunmuş olur.

$$L \subseteq (E_1 \cup E_2)^* \text{ için, } P_i(L) := \{ t \in E_i^* : \exists s \in L (P_i(s) = t) \} \quad (3.12)$$

$$L_i \subseteq E_i^* \text{ için, } P_i^{-1}(L) := \{ s \in (E_1 \cup E_2)^* : \exists t \in L_i (P_i(s) = t) \} \quad (3.13)$$

$P_i [P_i^{-1}] = L$ ’dir. Ama genellikle $L \subseteq P_i^{-1} [P_i]$ ’dir.

Örnek 3.3:

$E_1 = \{a, b\}$, $E_2 = \{b, c\}$ ve $L = \{c, ccb, abc, cacb, cabcbba\}$ ise,

$$P_1(L) = \{\varepsilon, b, ab, abba\}$$

$$P_2(L) = \{c, ccb, bc, ccbcb\}$$

$$P_1^{-1}(\{\varepsilon\}) = \{c\}^*$$

$$P_1^{-1}(\{b\}) = \{c\}^* \{b\} \{c\}^*$$

$$P_1^{-1}(\{ab\}) = \{c\}^* \{a\} \{c\}^* \{b\} \{c\}^*$$

$$P_1^{-1}[P_1(\{abc\})] = P_1^{-1}[\{ab\}] \supset \{abc\} \quad \text{olarak tanımlanır.}$$

Paralel kompozisyon ile elde edilen bir otomatın dilleri

$$\mathcal{L}(G_1 \parallel G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)] \quad (3.14)$$

$$\mathcal{L}_m(G_1 \parallel G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)] \quad (3.15)$$

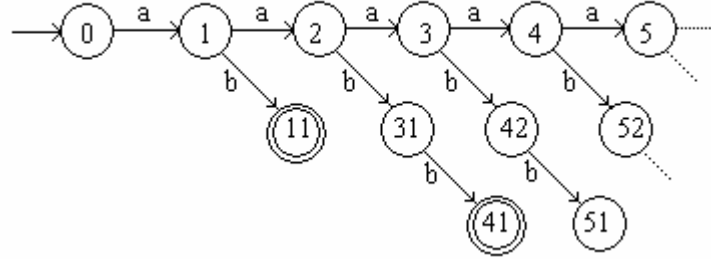
olarak tanımlanır.

4. SONLU DURUMLU OTOMATLAR

4.1 Regüler Dillerin Tanımı ve Özellikleri

Otomatlarda görevler belirli bir olaylar dizisinin oluşması ile tanımlanır ve bu olaylar dizisi bir “dil” oluşturur. Daha sonra bu dili üreten veya işaretleyen otomat tasarlanır. Ancak bu her zaman mümkün olmayabilir çünkü soyut bir problem için bir otomat tasarlanıyorsa durum uzayı sonsuz olabilir.

Şekil 4.1’de $L=\{a^n, b^n, \}$ ’de n ’lerin eşitliğini sağlayan bir sayaç yapısı kurulması gerekir. Bu yapı n sonsuza giderken durum uzayını da sonsuza götürür.



Şekil 4.1: $L=\{\epsilon, ab, aabb, aaabbb, \dots\}=\{a^n, b^n, : n \geq 0\}$ Dilini İşaretleyen Otomat

L ’nin kardinalitesinin sonsuz olması, sonsuz durumlu bir otomatla gösterilmesinin gerek koşulu değildir. E^* ’da kardinalitesi sonsuz olan bir dil, sonlu durumlu otomatla gösterilebilir.

Tanım 4.1 “Regüler Dil”:

Sonlu durumlu otomatlarla işaretlenen dil, “regüler dil” olarak tanımlanır, $\mathfrak{R}$ ile gösterilir. $\mathfrak{R}, 2^{E^*}$ kümesinin bir uygun alt kümesidir.

Regüler dillerin analiz ve kontrol sentezi problemlerini çözmek için otomatlar uygun bir formalizm oluşturur çünkü regüler dillerin durum uzayları sonludur ve programlama açısından sonlu hafızaya ihtiyaç duyarlar. Regüler olmayan diller ise sonsuz durum uzayına sahip oldukları için otomatlarla modellenmeleri uygun değildir. Petri Net formalizmi ise regüler olmayan bazı dilleri sonlu durum geçiş yapısı içinde modellemeye uygundur.

Belirgin olmayan sonlu durum bir otomata karşı düşen eşdeğer bir belirgin sonlu durum otomat vardır. Sonlu durum otomatlar belirgin olsun yada olmasın regüler diller ile “dual”dir.

Teorem 4.1 “Regüler Diller ve Sonlu Durum Otomatlar”:

Belirgin olmayan sonlu durum bir otomat ile gösterilebilen diller sınıfı, belirgin sonlu durum bir otomat ile gösterilebilen diller sınıfı $\mathfrak{R}$ ile aynıdır.

Teorem 4.2 “ $\mathcal{R}$ ’nin Özellikleri”: L_1 ve $L_2 \in \mathcal{R}$ ise aşağıdaki diller de regüler dildir.

$\overline{L_1}$ ön ek kapanışı,

L_1^* Kleen kapanışı,

$L_1^c = E^* \setminus L_1$ tümleyeni,

$L_1 \cup L_2, \quad L_1 L_2,$

$L_1 \cap L_2.$

4.2 Regüler İfadeler

$\mathcal{R}$ ’yi ifade etmenin bir diğer yolu da “regüler ifade”lerdir. Regüler ifadelerin tanımı için yeni bir notasyon kümesi belirlenmiştir.

- Birleşme “ $\cup$ ” yerine “ $+$ ”,
- Kleen kapanış $\{u\}^* = \{\epsilon, u, uu, uuu, \dots\}$ yerine “ u^* ”,
- $\{u\}$ ve $\{v\}$ türetmesiyle oluşan $\{uv\}$ yerine uv ,
- $\{u\}$ ve $\{v\}$ ’nin birleşimiyle oluşan $\{u,v\}$ yerine $(u+v)$,
- i. $\emptyset \in \mathcal{R}$, boş küme ile gösterilen regüler bir ifadedir.
 $\epsilon \in \mathcal{R}$, $\{\epsilon\}$ kümesi ile gösterilen regüler bir ifadedir.
 $E \in \mathcal{R}$, her $e \in E$ için $\{e\}$ kümesi ile gösterilen regüler bir ifadedir.
- ii. r ve s regüler ifadeler ise $rs, (r+s), r^*$ ve s^* de regüler ifadelerdir.
- iii. Tüm regüler ifadeler, i. ve ii. kuralların sonlu defa uygulanmasıyla oluşur.

Regüler ifadeler sonsuz sayıda kelimeye sahip karışık dillerin sonlu biçimde gösterilebilmesini sağlar.

Örnek 4.1 :

$E = \{a, b, g\}$ kümesi için $(a+b)g^*$ şeklinde verilen regüler ifade ile gösterilen dil L_1 ,
 $(ab)^*+g$ şeklinde verilen regüler ifade ile gösterilen dil L_2 ,

$$L_1 = \{a, b, ag, bg, agg, bgg, aggg, bggg, \dots\} \quad (4.1)$$

$$L_2 = \{\epsilon, g, ab, abab, ababab, \dots\} \quad (4.2)$$

olarak tanımlanır. L_1 ve L_2 'nin sonsuz sayıda elemanı olmasına rağmen karşı düşen regüler ifadeler basit sonlu bir ifadeye sahiptir.

Teorem 4.3 “Regüler İfadeler ve Regüler Diller”:

Regüler ifadelerle ile gösterilebilen diller regüler dillerdir, $\mathcal{R}$.

Bu teorem regüler ifadeler, regüler diller ve sonlu durum otomatların eşitliğini belirtir. Bu eşitlik “Kleene Teoremi” olarak adlandırılır.

5. OTOMATLAR VE PETRİ AĞLARININ KARŞILAŞTIRILMASI

Her iki formalizm de sonlu durum sistemlerinin davranışlarını modellemek için kullanılırlar ve geçiş yapılarını açık olarak ifade ederler.

Otomatlarda bu yapı, olası bütün durumların numaralandırılması ve bu durumların olası bütün geçişlerle bağlanması sonucunda oluşturulur. Böylece yapıda geçiş fonksiyonları da belirtilmiş olur. Otomatlar sadece görsel açıdan değil aynı zamanda çarpma ve paralel kompozisyon gibi işlemlerle kolayca birleştirilebildiği için karmaşık sistemlerin parçalar halinde modellenmesini kolaylaştırır. Petri ağlarında ise geçiş fonksiyonlarının yapıları daha fazla öne çıkarılmıştır. Durumlar numaralandırılmamış, geçişler sonrasında ortaya çıkacak şekilde sistem içinde dağıtılmıştır.

Verilen bir sonlu durum sistemin modellenmesinde hangi modelin daha iyi olduğunu sorusunun aşık bir cevabı yoktur, sistemin yapısına ve kişilerin tercihlerine göre değişebilir. İki modelin karşılaştırılması için belirli kriterler belirlenmiştir. Ancak Petri ağları ve otomatlar birbirleriyle rekabet eden değil, birbirlerini tamamlayan modelleme yöntemleridir.

5.1 Dilin İfade Edilebilirliği

Otomatlar ve Petri ağlarının karşılaştırılmasındaki ilk kriter, her iki formalizmin gösterebildiği dillerdir çünkü sonlu hafıza pratik bir sınırlamadır. PNL, $\mathcal{R}$ 'den daha geniş bir sınıfa sahiptir. E^* 'da sonlu durum ve sonlu geçişli Petri ağları ile ifade edilen dillerin sayısı sonlu durumlu otomatlarla ifade edilen dillerin sayısından daha

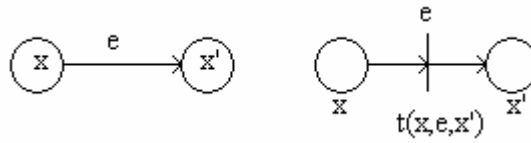
fazladır. Bunu ispat edebilmek için önce bir sonlu durum otomatın her zaman aynı dili işaretleyen ve üreten bir Petri ağına çevrilebileceği daha sonra regüler olmayan bir dilin Petri ağı ifade edilebildiğini bulmak yeterli olacaktır.

$G = (X, E, f_g, \Gamma, x_0, X_m)$ sonlu durum bir otomatta X ve E de sonlu birer küme olacaktır. $N = (P, T, A, \omega, E, l, \Gamma, x_0, X_m)$ Petri ağını,

$$\ell(G)=\ell(N) \text{ ve } \ell_m(G)=\ell_m(N) \quad (5.1)$$

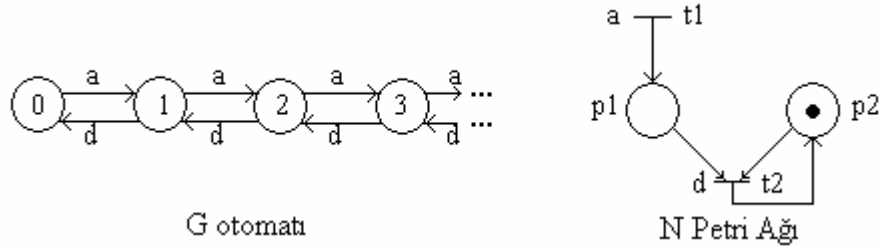
koşullarını sağlayacak şekilde oluşturmak için;

- i. X kümesindeki her duruma karşılık P kümesinde bir yer tanımlanır. ($P=X$)
 - N 'deki x_0 , G 'deki $x_0 \in X$ durumuna karşı düşen yer, 1 olacak şekilde $[0, \dots, 0, 1, 0, \dots, 0]$ olarak tanımlanan bir vektördür.
 - N 'deki x_m kümesi, G 'deki $x_m \in X$ durumuna karşı düşen yer 1 olacak şekilde $[0, \dots, 0, 1, 0, \dots, 0]$ olarak tanımlanan vektörler kümesidir.
- ii. G 'deki (x, e, x') üçlüsü, $e \in \Gamma(x)$ aktif olay kümesindeki e 'ler için $x' = f_g(x, e)$ şeklinde gösterilir. N 'de ise bu geçişler $t(x, e, x') \in T$ şeklinde olacaktır. Diğer bir deyişle T 'nin kardinalitisi geçiş diyagramındaki oklar kümesinin kardinalitisiyle aynı olacaktır.
 - T 'deki (x, e, x') geçişleri $e \in E$ olayları ile etiketlenir.
 - G 'deki her (x, e, x') üçlüsü için, A 'da ağırlıkları '1' olacak şekilde iki ark tanımlanır, $\text{arc}(x, t(x, e, x'))$ ve $\text{arc}(t(x, e, x'), x)$.



Şekil 5.1: G 'deki (x, e, x') Üçlüsünün N 'deki Karşılığı.

Regüler olmayan bir dilin Petri ağı ifade edilebildiğini göstermek için;



Şekil 5.2 G otomatının, N Petri Ağı Eşleniği

Şekil 5.2’de tanımlanan N Petri ağında $x_0 = [0,1]$ ve $X_m = \{[x, y]: x \in \mathbb{N} \text{ ve } y \in \{0,1\}\}$, G otomatında $x_0 = 0$ ve $X_m = \mathbb{N}$ olarak alınırsa, N Petri ağı regüler olmayan G otomatıyla aynı dili $\{ad\}^*$ ’yi işaretler. G’de a ve d olaylarının eşit sayıda olmasını sağlamak için bir hafızaya ihtiyaç duyulur ve bu otomatın durum sayısını oldukça büyütüp sonsuza götürebilir. Ancak N’de böyle bir problem oluşmaz çünkü sonlu durum ve sonlu geçişe sahiptir.

Bütün Petri ağlarına karşı düşen bir otomat bulunması mümkün olmazken, erişilebilir durumlarının sayısı $(R(N))$ sonlu olan her Petri ağına karşı düşen bir otomat bulunabilir. Bir Petri ağının erişilebilir durumlarının sayısı sonsuz olabildiği için bir otomatın işaretlediği yada üretebildiği dilden daha geniş bir dil ailesini işaretler.

5.2 Modüler Model İnşa Etme

Her ne kadar basit sonlu durumlu bir otomata karşı düşen Petri ağı karmaşık bir yapıya sahip olabilse de Petri ağlarının bazı yapısal avantajlara sahiptir. Bunlardan biri karmaşık bir sistemi modülerize etme ve alt sistemlerine kolayca ayrıştırılabilmesidir.

Etkileşimli olarak çalışan X_1 durumlu bir otomat ile X_2 durumlu başka bir otomat bağlandığı zaman durum uzayı sayısı $X_1 \times X_2$ ’ye eşit olur. Bu yeni elde edilen otomatın durum sayısının aşırı artması yani sistemin karmaşıklaşması anlamına gelir. Diğer taraftan Petri ağları modelinde bu bağlantıyı yapmak daha kolaydır, sisteme birkaç yer ve/veya geçiş ekleyerek yada birkaç yeri birleştirerek yapılabilir. Bu da durum sayısının üssel değil lineer olarak artmasına sebep olur.

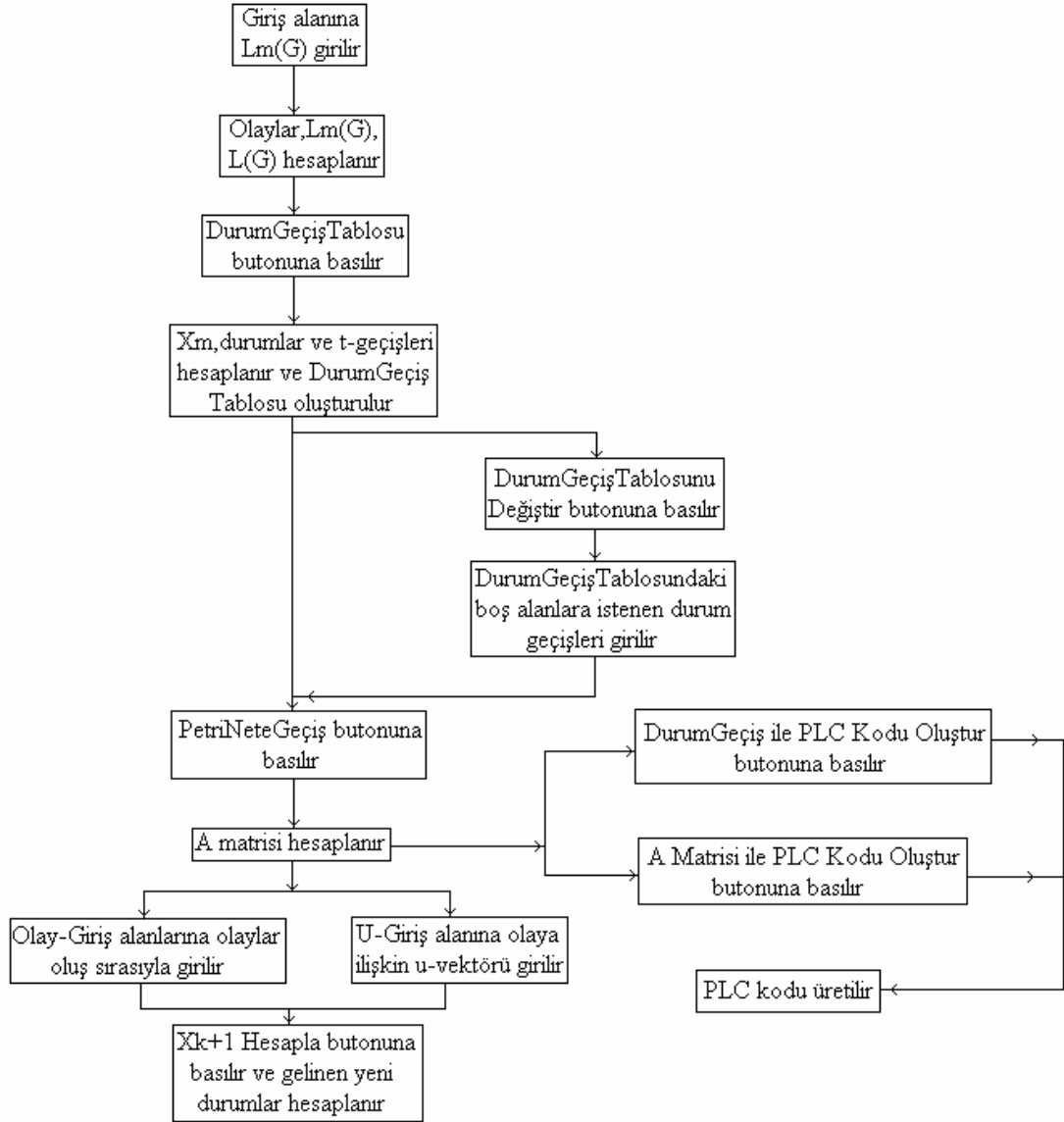
5.3 Karar Verilebilirlik

Bir olay dizisi sonlu durum bir otomat tarafından tanınabilir mi? Karar verilebilirlik, bu soruya evet yada hayır diyebilmenin sistematik bir yolu olup olmaması olarak ele alınır. Sonlu durum otomatlar her zaman “karar verilebilir” bir yapıya sahiptir. Petri ağları ise model zenginliğine sahip oldukları için “karar verilebilir” bir yapıya sahip olmayabilirler.

6. PROGRAM AÇIKLAMALARI

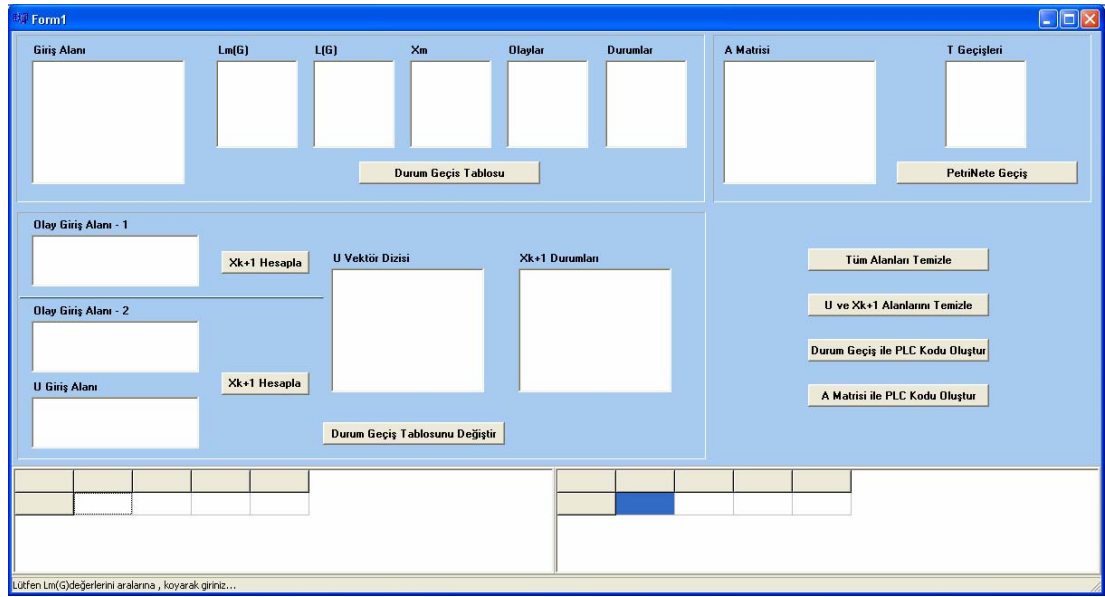
Bu çalışmada, biçimsel işaretli bir dilden endüstriyel bir işlemciye otomatik kod üreten bir program geliştirme amaçlanmıştır. Bu amaç için, önce işaretli dilden üretilen dile geçilmiş ve bu dili üreten otomatın durum geçiş diyagramı elde edilmiştir. Daha sonra durum geçiş matrisinden sistemin Petri ağ modeli elde edilmiş ve bu model temel alınarak kullanılan işlemciye uygun kod üreten yazılım gerçekleştirilmiştir. Bu işlemler, tüm aşamaları görsel olarak sunmaya imkan veren Borland C++ Builder ortamında gerçekleştirilmiştir. İzlenen yöntemle ilişkin işaret akış diyagramı Şekil 6.1’de gösterildiği gibidir.

Endüstriyel işlemci olarak SIMATIC-300 seçilmiş ve standart bir dil olan SCL dilinde kod üretilmiştir. Benzer yazılımlarda ortaya çıkan ve ilgili literatürde “çığ etkisi” (avalanche effect) olarak adlandırılan problem analiz edilmiş ve çığ etkisinden arındırılmış SCL kodu üretilmiştir. İzlenen yöntemle ilişkin işaret akış diyagramı Şekil 6.3 ve Şekil 6.4 ’de gösterildiği gibidir.



Şekil 6.1 : Programa İlişkin Akış Diyagramı

6.1 Borland C++ Builder Programı



Şekil 6.2 : Program Arayüzü

Program, “Giriş Alanı”na işaretli dilin (Lm(G)) girilmesiyle başlar. Lm(G)’ye ait kelimeler aralarına virgöl konularak yazılmalıdır. Giriş alanına (GirisMemo) ilişkin “GirisMemoKeyPress” ve “GirisMemoKeyDown” fonksiyonları vardır. GirisMemoKeyPress fonksiyonunda giriş alanına sayı girilmesi, space ve back tuşlarına basılması engellenir. GirisMemoKeyDown fonksiyonunda girilen her kelimenin önek kapanışları ile tüm alt kelimeleri, kelimeyi oluşturan harfler ile de olaylar belirlenir. Girilen kelimeler “LmListBox”a, alt kelimeler “LListBox”a ve olaylar “EventListBox”a eklenir.

```
void __fastcall TForm1::GirisMemoKeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    // Giriş alanına girilen Lm(G)'den L(G) ve olayları elde ediyoruz.
    int text_len, tuple_len, i, j, len;
    AnsiString LmTuple, LTuple, Event;
    LmTuple = "";
    LTuple = "";
    /* Lm(G) de girilen her kelime ',' (Key == 188) ile birbirinden ayrılır.
       Enter tuşuna ya da ',' e her basıldığında yeni bir kelime girilmiş
       demektir. Enter tuşundan sonra Giriş alanına yeni bir kelime girilemez.
    */
    if((Key == 188) || (Key == VK_RETURN))
    {
        /* Girilen kelimenin uzunluğunu hesaplıyoruz. Kelimeler arka
           arkaya girildiği için text_len bu ana kadar girilen Lm(G)'nin
           toplam uzunluğudur.
        */
        text_len = GirisMemo->Text.Length();
```

```

/* Global bir deęişken olan string_len'de bir önceki
kelimenin bittięi indeks tutulur. Böylece (string_len+1)
den text_len'e kadar olan kısım yeni girilen kelimedir.
Eđer yeni girilen kelimenin uzunluęu sıfır deęilse ilgili
tablolarla gerekli eklemeler yapılır.
*/
if((text_len != 0)&&((text_len-string_len)!=0))
{
    // Yeni girilen kelime LmTuple da saklanır.
    LmTuple = GirişMemo->Text.SubString(string_len+1,text_len-string_len);
    // Yeni kelime Lm(G) ye eklenir.
    LmListBox->Items->Add(LmTuple);
    tuple_len = LmTuple.Length();
    /* Yeni kelimedeki her olay (harf) tek tek EventListBox'da
    aranır. Eđer daha önce eklenmediyse eklenir.
    */
    for(i = 1; i<tuple_len+1;i++)
    {
        Event = LmTuple.SubString(i,1);
        for(j = 0; j<EventListBox->Items->Count; j++)
        {
            if(Event == EventListBox->Items->Strings[j])
                break;
        }
        if(j == EventListBox->Items->Count)
            EventListBox->Items->Add(Event);
    }
    /* Yeni kelimenin tüm alt kelimeleri hesaplanır (LTuple)
    ve L(G)'ye (LListBox) a daha önceden eklenmediyse eklenir.
    */
    for(i = 1; i<tuple_len+1;i++)
    {
        LTuple = LmTuple.SubString(1,i);
        for(j = 0; j<LListBox->Items->Count; j++)
        {
            if(LTuple == LListBox->Items->Strings[j])
                break;
        }
        if(j == LListBox->Items->Count)
            LListBox->Items->Add(LTuple);
    }
    // string_len güncellenir. +1 aralara girilen virgül için konulmuştur.
    string_len = text_len + 1;
}
// Enter'a basıldığında Giriş alanını deaktif ediyoruz.
// Yeni bir Lm(G) girmek için 'Tüm Alanları Temizle' tuşuna basılmalı.
if(Key == VK_RETURN)
{
    GirişMemo->Enabled = false;
}
}
//-----
void __fastcall TForm1::GirişMemoKeyPress(TObject *Sender, char &Key)
{
    // Giriş alanında sayı girilmesine, space ve back tuşlarının basılmasına izin vermiyoruz.
    if((Key == VK_SPACE)||((Key >= '0')&&(Key <= '9'))||(Key == VK_BACK))
        Key = 0x0;
}

```

Daha sonra “Durum Geçiş Tablosu” tuşuna basılmalıdır. Bu tuşa basıldığında “DurumGecisBitBtnClick” fonksiyonu çalışır ve durum geçiş tablosu “DGStringGrid”, durumlar ve işaretli durumlar hesaplanır. Durum geçiş tablosunun 0.satır ve 0.sütunu etiketler için ayrılmıştır. 0. satıra olaylar 0. sütuna durumlar sırasıyla yazılır. Olayların yer alacağı sütun sayısı olay sayısının bir fazlası, durumların yer alacağı satır sayısı durum sayısının bir fazlası kadardır. Otomatın durumları “1”den başlayarak sırayla numaralandırılır, “1” başlangıç durumu olarak kabul edilir ve “StateListBox”a ve “XmListBox”a eklenir. Lm(G)’deki her kelime bir çevrim içerisinde alt kelimelerine bölünür ve bu alt kelimeler daha önce StateListBox’a eklenmediyse yeni bir durum oluşmuş demektir. Oluşan yeni duruma karşı düşen alt kelime durum numarasıyla birlikte StateListBox’a eklenir. Durum geçiş tablosu, DGStringGrid[initial_state, oluşan_olay]=last_state olacak şekilde güncellenir. Eğer alt kelime StateListBox’a daha önce eklendiyse bu alt kelimeye karşı düşen durum initial_state’e atanır ve böylece bir sonraki çevrim bu durumdan devam eder. Lm(G)’ye ait her kelimenin son harfi yani olayı otomatı işaretli bir duruma götürür. Bu durumlar “XmListBox”a eklenir. Durum değişikliğine sebep olan her olay bir t-geçişidir. x. t-geçışı, (x+1). duruma geçilmesini sağlayacak şekilde t-geçişleri ve bunlara ilişkin olaylar “TransListBox”a eklenir. Petri ağına geçişi sağlamak amacıyla “PNStringGrid” tablosu oluşturulur. Bu tablonun satır sayısı durum sayısının bir fazlası, sütun sayısı ise t-geçişleri sayısının bir fazlası kadardır. Durum geçiş tablosundan tek farkı sütunlarında olaylar yerine t-geçişlerinin bulunmasıdır. Tablonun içi PNStringGrid[initial_state, t-geçışı]=last_state olacak şekilde güncellenir.

```
// 'Durum Geçiş Tablosu' tuşuna basıldı.
void __fastcall TForm1::DurumGecisBitBtnClick(TObject *Sender)
{
    int i,j,col_no, len;
    initial_state = 1;
    last_state = 0;
    int lm_len = 0;
    int tseparator = 0;
    int idle_state;

    AnsiString LmTuple,Lm,State,T_Event,T_State,T_String,TIndex;

    // Durum Geçiş matrisinin 0.satır ve 0.sütunu etiketler için ayrılmıştır.
    // Bu durumda olayların yer alacağı sütun sayısı olay sayısı + 1 'dir.
    DGStringGrid->ColCount = EventListBox->Items->Count+1;

    // Durum Geçiş matrisinin 0.satırına olaylar yazdırılır.
    for(i=0;i<EventListBox->Items->Count; i++)
    {
        DGStringGrid->Cells[i+1][0] = EventListBox->Items->Strings[i];
    }
    // Başlangıç durumu, durumlar alanına yazdırılır.
    StateListBox->Items->Add("1-initial state");
```

```

// Durum Geçiş matrisinin 1.satırının 0.sütununa ilk durum '1' yazdırılır.
DGStringGrid->Cells[0][1] = "1";

//Başlangıç durumu XMLListBox'a eklenir.
XmListBox->Items->Add("1");

/* Lm(G) deki kelimeler 'LmTuple' teker teker bulunur ve Kleen
   Kapanış'ları hesaplanarak Durum Geçiş Matrisi oluşturulur.
*/
for(i=0;i<LmListBox->Items->Count; i++)
{
    len = LmListBox->Items->Strings[i].Length();

    LmTuple = LmListBox->Items->Strings[i];

    for(j=1;j<len+1;j++)
    {
        // LmTuple'nin her alt kelimesi yani Kleen kapanışı
        // döngü içinde hesaplanır ve Lm'ye atanır.
        Lm = LmTuple.SubString(1,j);

        // Lm'i oluşturan olaya (Lm'in son harfi) karşılık düşen sütun numarası bulunur.
        col_no = find_column_no(Lm,j);

        if(col_no == 0) return;

        // Durum Geçiş Tablosu oluşturulurken ağaç yapısı kullanılır.
        // Eğer eklenmek istenen Lm daha önceden eklenmişse, yani ağaçta
        // bu Lm'e karşı düşen bir durum var ise initial_state bu durumla
        // güncellenir. Eğer yoksa bu bizi yeni bir duruma götürecektir.
        //
        if(!Form1->check_state_listbox(Lm))
        {
            // Yeni bir durum oluştu, last_state ve Durum Geçiş Matrisinin satır sayısı bir arttırılır.
            last_state++;
            DGStringGrid->RowCount++;

            // Yeni durum, Durum Geçiş Tablosu'na eklenir.
            DGStringGrid->Cells[col_no][initial_state] = last_state+1;

            // initial_state güncellenir.
            initial_state = last_state+1;

            // Durum listesi güncellenir. Yeni duruma karşılık düşen kelime araya '-' koyarak yazılır.
            State = IntToStr(last_state+1)+"-"+Lm;
            StateListBox->Items->Add(State);

            //İşaretili durumlar için XmListBox güncellenir.
            if(j == len)
            {
                XmListBox->Items->Add(IntToStr(last_state+1));
            }

            // Petri Nete Geçişte kullanılmak üzere olaylara karşı düşen 't' geçişleri hesaplanır.
            lm_len = Lm.Length();
            T_Event = Lm.SubString(lm_len,1);
            T_State = "t"+IntToStr(last_state);
            T_String = T_State + "-" + T_Event;
            TransListBox->Items->Add(T_String);

            /* Eklenen yeni durum 0.sütuna etiket olarak yazılır.
               Durumlar 1.satırdan itibaren yazılmaya başladığı için +1 konulmuştur.*/
            DGStringGrid->Cells[0][last_state+1] = last_state+1;
        }
    }
}

```

```

for(i=1;i<DGStringGrid->RowCount; i++)
{
    for(j=1;j<DGStringGrid->ColCount; j++)
    {
        if(DGStringGrid->Cells[j][i] != "")
            break;
    }
}

/* Yığılma problemini önlemek için, Durum Geçiş Tablosu Petri Net geçişine
uygun hale getirilir. Sütunlara karşılık düşen olaylar yerine
yukarıda hesaplanan 't' geçişleri kullanılır.
*/

/* Yeni tablonun sütun sayısı 't' lerin sayısı +1 dir, satırlarının
sayısı ise Durum Geçiş Matrisi ile aynıdır, yani durum sayısı +1 dir.
*/
PNStringGrid->ColCount = TransListBox->Items->Count+1;
PNStringGrid->RowCount = DGStringGrid->RowCount;

// Yeni matrisin 0.sütunlarına 1. satırdan başlayarak durumlar yazılır.
for(i=1;i<PNStringGrid->RowCount; i++)
{
    PNStringGrid->Cells[0][i] = IntToStr(i);
}
// Yeni matrisin 0.satırına 1. sütundan başlayarak 't' ler yazılır.
for(i=0;i<TransListBox->Items->Count; i++)
{
    tseparator = TransListBox->Items->Strings[i].Pos("-");

    if(tseparator > 1)
        TIndex = TransListBox->Items->Strings[i].SubString(1,tseparator-1);

    PNStringGrid->Cells[i+1][0] = TIndex;
}

/* Durum Geçiş Matrisinin hesaplanma şekline göre bizi i.durumdan
X durumuna (DGStringGrid->Cells[j][i]) götüren t, t(X-1) yani
t(DGStringGrid->Cells[j][i])-1 dir. Bu durumda yeni matrisin
i.satırının t(X-1).sütununa X yazılır.
*/
for(i=1;i<DGStringGrid->RowCount; i++)
{
    for(j=1;j<DGStringGrid->ColCount; j++)
    {
        if(DGStringGrid->Cells[j][i] == "")
            continue;

        PNStringGrid->Cells[(DGStringGrid->Cells[j][i]-1)][i] = DGStringGrid->Cells[j][i];
    }
}

// Durum Geçiş Matrisinden son durumlar belirlenir.
find_final_states();
}

//-----

// Durum Geçiş matrisinin sütunlarını, EventListBox'a eklendikleri sıra ile
// olaylar oluşturur. Satırları ise numara sırasıyla durumlar oluşturur.
// 0.satır ve 0. sütunlar da olay ve durum etiketleri için ayrılmıştır.
int find_column_no(AnsiString Lm,int len)
{
    AnsiString LmLast;
    int i;

```

```

// Lm deki kelimelerin tüm alt kelimeleri için bu işlem tekrarlandığından
// son harfe yani olaya bakmak yeterlidir.
LmLast = Lm.SubString(len,1);

/* Olayın EventListBox daki indeksi 'i' bulunur. Durum Geçiş Matrisinde
   olaylar 1.sütundan başladığı için 'i+1' döndürülür.
*/
for(i=0;i<Form1->EventListBox->Items->Count; i++)
{
    if(Form1->EventListBox->Items->Strings[i] == LmLast)
        return(i+1);
}

if(i == Form1->EventListBox->Items->Count) return(0);
}

//-----

bool __fastcall TForm1::check_state_listbox(AnsiString Lm)
{
    int i,separator,length;
    AnsiString StateTuple,MainTuple;
    /* Alt kelime Lm, StateListBox da aranır. Ancak statelere karşılık düşen
       kelimeler '-' den sonra yazıldığı için MainTuple hesaplanır.
    */
    for(i=1;i<Form1->StateListBox->Items->Count;i++)
    {
        StateTuple = Form1->StateListBox->Items->Strings[i];

        length = StateTuple.Length();
        separator = StateTuple.Pos("-");

        MainTuple = StateTuple.SubString(separator+1,length-separator);
        /* Eğer alt kelime Lm'e karşılık düşen bir durum varsa
           initial_state bu değerle güncellenir.
        */
        if(Lm == MainTuple)
        {
            initial_state = StrToInt(StateTuple.SubString(1,separator-1));
            return(true);
        }
    }
    /* Eğer alt kelime Lm'e karşılık düşen bir durum yoksa ve bu ilk alt
       kelimeyse yani uzunluğu 1 ise initial_state'e 1 atanır. Ağaç yapısına
       yeni bir dal eklenir.
    */
    if(Lm.Length() == 1)
        initial_state = 1;

    return(false);
}

//-----

// Durum Geçiş Matrisinden son durumlar belirlenir.
void __fastcall TForm1::find_final_states()
{
    int i,j, counter;

    // Son durumlar sıfırlanır.
    for(i=0;i<100; i++)
    {
        final_states[i] = 0 ;
    }
}

```

```

counter = 0;

// Eğer bir durum hangi olay olursa olsun durum değiştirmiyorsa (Durum Geçiş
// Matrisinde bu duruma karşılık gelen sütunların hepsi sıfırsa) "son durum"
// yani kilitlenmenin olduğu durum olarak kabul edilir.
for(i=1;i<DGStringGrid->RowCount; i++)
{
    for(j=1;j<DGStringGrid->ColCount; j++)
    {
        if(DGStringGrid->Cells[j][i] != "")
            break;
    }

    // Son durumlar 0. indeksten başlayarak final_states'e kaydedilir.
    if (j == DGStringGrid->ColCount )
    {
        final_states[counter] = i;
        counter = counter +1;
    }
}
}

```

Şimdiye kadar yapılan çalışmalar sadece sonlu bir dili desteklemektedir. Sonsuz dile geçiş için Durum Geçiş Tablosunun (DGStringGrid) boş alanlarına yeni durum geçişlerinin eklenmesine izin verilir. “Durum Geçiş Tablosunu Değiştir” butonuna basılarak DGStringGrid yazılabilir hale getirilir. Dolu alanların değiştirilmesine izin verilmezken, boş alanlara istenen durumlar eklenebilir. Eklenen durum, var olan durumlar arasından seçilmeli ve ENTER’a basılarak giriş yapılmalıdır. Her yeni giriş için PNStringGrid tablosu bu yeni girişe ilişkin t-geçişini içerecek şekilde güncellenir. Aynı zamanda t-geçişlerini listeyen TransListBox kutusuna yeni geçişler eklenir.

```

void __fastcall TForm1::bbEditClick(TObject *Sender)
{
    // "DG Değiştir" tuşuna basıldığında Durum Geçiş Tablosu yazılabilir hale getirilir.

    DGStringGrid->Options.operator <<(goEditing);
}

void __fastcall TForm1::DGStringGridKeyPress(TObject *Sender, char &Key)
{
    int place_count,event_count,trans_count;
    AnsiString T_String,T_State,T_Event;

    place_count = DGStringGrid->RowCount-1;

    event_count = DGStringGrid->ColCount-1;

    //Durum Geçiş Tablosu'na Boşluk, Harf ve Durum Sayısından büyük tek rakam girişleri engellenir.

    if(Key != VK_RETURN)
    {
        if((Key == VK_SPACE)||((Key <= '0')||(Key > '9'))||(AnsiString(Key) > IntToStr(place_count)))
        {

```

```

        Key = 0x0;
        return;
    }
}

//ENTER tuşuna basıldığında yeni t geçişi PNStringGrid tablosuna ve TransListBox'a eklenir.

if(Key == VK_RETURN)
{
    if(DGStringGrid->Cells[selected_col][selected_row] != "")
    {
        trans_count = TransListBox->Items->Count;

        T_State = "t"+IntToStr(trans_count+1);
        T_Event = DGStringGrid->Cells[selected_col][0];

        T_String = T_State + "-" + T_Event;
        TransListBox->Items->Add(T_String);

        trans_count++;

        PNStringGrid->ColCount++;

        PNStringGrid->Cells[trans_count][0] = "t"+IntToStr(trans_count);

        PNStringGrid->Cells[trans_count][selected_row] = DGStringGrid->Cells[selected_col][selected_row];

        return;
    }
}

void __fastcall TForm1::DGStringGridSelectCell(TObject *Sender, int ACol,
int ARow, bool &CanSelect)
{
    int trans_count;
    AnsiString T_String,T_State,T_Event;

    //Durum Geçiş Tablosu'nda dolu olan alanların seçimi engellenir.

    if(DGStringGrid->Cells[ACol][ARow] != "")
    {
        CanSelect = false;
        return;
    }

    selected_col = ACol;
    selected_row = ARow;
}

void __fastcall TForm1::DGStringGridSetEditText(TObject *Sender, int ACol,
int ARow, const AnsiString Value)
{
    int place_count;

    place_count = DGStringGrid->RowCount-1;

    //Durum Geçiş Tablosu'nda giriş yapılan değerin durum sayısından büyük olması engellenir.

    if(Value != "")
    {
        if(StrToInt(Value) > place_count)
            DGStringGrid->Cells[ACol][ARow] = "";
    }
}

```

Daha sonra “PetriNete Geçiş” tuşuna basılmalıdır. Bu tuşa basıldığında “PetriNetBitBtnClick” fonksiyonu çalışır ve A matrisi, $a[j][i] = t_p[j][i] - p_t[j][i]$ olacak şekilde hesaplanır. Burada j her zaman t-geçiş indeksi, i ise p yeri indeksidir. Eğer PNStringGrid’de j. sütunu ve i. satırına ait bir geçiş tanımlı ise yani $PNStringGrid[j][i] = \text{yeni_p} \neq ""$ ise, $p_t[j][i]$ ve $t_p[j][\text{yeni_p}]$ “1” değerini alır. Tüm p_t ve t_p değerleri hesaplandıktan sonra A matrisi yukarıdaki denkleme göre hesaplanır. Her olaya ilişkin t-geçiş sayısı olay ismi ile birlikte “EListBox”a eklenir. Bu liste $x_{k+1} = x_k + u_k A$ hesaplanırken kullanılmak üzere tutulmaktadır, ekranda görülmez.

```
// 'PetriNete Geçiş' tuşuna basıldı.
void __fastcall TForm1::PetriNetBitBtnClick(TObject *Sender)
{
    int trans_count= 0;    //row_count
    int place_count= 0;    //col_count

    int t_p[100][100],p_t[100][100];
    AnsiString A_Row = "";
    // Önce Durum Geçiş Tablosu hesaplanmalı.
    if(PNStringGrid->Cells[0][1] == "")
    {
        Application->MessageBox("Önce Durum Geçiş Tablosunu oluşturunuz!", "Uyarı", MB_OK);
        return;
    }

    // Durumların sayısı Durum Geçiş Matrisinin satır sayısının 1 eksiğine,
    // geçiş sayısı TransListbox'ın eleman sayısına eşittir.
    place_count = DGStringGrid->RowCount - 1;
    trans_count = TransListBox->Items->Count;

    // Matris çarpımında kullanılan t_p, p_t ve a matrisleri sıfırlanır.
    for(int i=0;i<100;i++)
    {
        for(int j=0;j<100;j++)
        {
            a[i][j] = 0;
            t_p[i][j] = 0;
            p_t[i][j] = 0;
        }
    }

    //A matrisi,  $a[j][i] = t_p[j][i] - p_t[j][i]$  kuralına göre hesaplanır.
    for(int i=1;i<place_count+1;i++)
    {
        for(int j=1;j<trans_count+1;j++)
        {
            if(PNStringGrid->Cells[j][i] != "")
            {
                p_t[j][i] = 1;
                t_p[j][StrToInt(PNStringGrid->Cells[j][i])] = 1;
            }
        }
    }

    for(int i=1;i<place_count+1;i++)
    {
        for(int j=1;j<trans_count+1;j++)
        {
```

```

        a[j][i] = t_p[j][i]-p_t[j][i];
    }
}

// A matrisi ekrana yazdırılır.
for(int j=1;j<trans_count+1;j++)
{
    A_Row = "";

    for(int i=1;i<place_count+1;i++)
    {
        if(i == 1)
            A_Row = IntToStr(a[j][i]);
        else
            A_Row = A_Row + " " + IntToStr(a[j][i]);
    }

    AListBox->Items->Add(A_Row);
}

// Olaylar sayıları ile birlikte EListBox'a eklenir.
// (xk+1 i A matrisinden hesaplamak için)

calc_event_count();
}
//-----
// Olay Giriş Alanı 1'e girilen her olayı t geçişi sayısı ile birlikte EListBox'a ekler.
void __fastcall TForm1::calc_event_count()
{
    int tseparator,event_count;
    AnsiString Event,EventString;

    EListBox->Items->Clear();

    Event = "";
    EventString = "";

    for(int j=1;j<DGStringGrid->ColCount;j++)
    {
        Event = DGStringGrid->Cells[j][0];

        event_count = 0;

        for(int i=0;i<TransListBox->Items->Count;i++)
        {
            tseparator = TransListBox->Items->Strings[i].Pos("-");

            if(Event == TransListBox->Items->Strings[i].SubString(tseparator+1,1))
                event_count++;
        }

        EventString = Event + "-" + IntToStr(event_count);

        EListBox->Items->Add(EventString);
    }
}

```

Durum geçiş tablosu ve A matrisi hesaplandıktan sonra $x_{k+1} = x_k + u_k A$ denklemi kullanılarak girilen olaylar ya da u giriş vektörlerine göre yeni durumlar belirlenir. Bunun için üç farklı yöntem kullanılmıştır. Başlangıç durumu olarak $X_0 = [1, 0, \dots, 0]$ alınmıştır.

- “Olay Giriş Alanı – 1” kullanılarak: “U1AnilMemo” alanına ilişkin “U1AnilMemoKeyDown” ve “U1AnilMemoKeyPress” fonksiyonları tanımlıdır. U1AnilMemoKeyPress fonksiyonunda giriş alanına sayı girilmesi ve space tuşuna basılması engellenir. Olayların tek tek ve aralarına virgöl konularak girilmesi sağlanır. İşlem bittiğinde ya ENTER tuşuna basılmalı veya virgöl konulmalıdır. U1AnilMemoKeyDown fonksiyonunda, girilen her olaya ait t geçişleri kullanılarak, tüm olası u vektörleri belirlenir ve “U Vektör Alanı”na, “UListBox”, eklenir. Her eklenen vektöre karşı düşen olay ise “EListBox1” listesine yazılır. Bu liste $x_{k+1} = x_k + u_k A$ hesaplanırken kullanılmak üzere tutulmaktadır, ekranda görülmez.

U1AnilMemo alanına girilen her olay için UListBox da olayın t-geçişleri sayısı kadar u vektörü bulunur. U1AnilMemo alanına giriş tamamlandığında bu alanın yanındaki “X_{k+1} Hesapla” tuşuna basılmalıdır. Bu tuşa basıldığında “AnilXkBitBtnClick” fonksiyonu çalışır. Bu fonksiyonda UListBox’taki her u vektörü için $X_k[j] = X_o[j] + U A[j]$ değeri hesaplanır, eğer elde edilen durum vektörünün herhangi bir elemanı sıfırdan küçükse sistem bu u girişi için durum değiştirmez ve sistem durumunu korur ($X_k = X_o$). Aksi halde belirlenen yeni durum “Xk”, bir önceki durumdan “Xo”, farklı ise sistem durum değiştirmiş demektir. Yeni durum Xo’a atanır ($X_o = X_k$). Sistem durum değiştirsin ya da değiştirmesin yapılan her hesaplama için Xk durum vektörü “X_{k+1} Durumları” alanına eklenir. Durum değişikliğine sebep olan u vektörünün ve Xk yeni durum vektörünün yanına “*” işareti konulur. Her olay için bir t-geçiş durumu değişikliğine sebep olacağından sistem durum değiştirdiğinde o olaya ilişkin diğer u vektörleri atlanır, bir sonraki olaya ait u vektörlerine EListBox ve EListBox1 listeleri kullanılarak gidilir. Hesaplamalar burdan devam eder. Eğer sistem açmaz bir duruma ulaşırsa başlangıç durumuna döndürülür.

```
// U1AnilMemo alanına olaylar aralarına virgöl konularak sirayla girilir.
// t geçişleri kullanılarak girilen olaylara karşı düşen tüm u vektörleri belirlenir.
void __fastcall TForm1::U1AnilMemoKeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    int tseparator, text_len, t_num, tlen;

    AnsiString UTuple, MainU, TEvent;

    UTuple = "";
    MainU = "";
    TEvent = "";

    t_num = 0;
    text_len = 0;
```

```

tseparator = 0;
tlen = 0;

// Virgül ya da entera basılırsa
if((Key == 188)|| (Key == VK_RETURN))
{
    if(DGStringGrid->Cells[1][0] == "")
    {
        Application->MessageBox("Lütfen Durum Geçiş Tablosunu
                                oluşturunuz!", "Uyarı", MB_OK);
        U1AnilMemo->Text = "";
        return;
    }

    int place_count = DGStringGrid->RowCount - 1;
    int trans_count = TransListBox->Items->Count;

    if(AListBox->Items->Count == 0)
    {
        Application->MessageBox("Lütfen Önce PetriNet'e Geçiniz!", "Uyarı", MB_OK);
        U1AnilMemo->Text = "";
        return;
    }

    // U1AnilMemo alanına olaylar tek tek ve aralarına virgül
    // konularak girilir, bu yüzden virgül ya da entera her basıldığında
    // U1AnilMemo'daki verinin son harfi girilen olaya, MainU'ya, eşittir.
    text_len = U1AnilMemo->Text.Length();
    MainU = U1AnilMemo->Text.SubString(text_len, 1);

    // Girilen olay (MainU) için t geçişleri kullanılarak, tüm olası u vektörleri
    // belirlenir ve U vektör dizisine eklenir.
    for(int i=0; i<TransListBox->Items->Count; i++)
    {
        tlen = TransListBox->Items->Strings[i].Length();
        UTuple = "";
        tseparator = TransListBox->Items->Strings[i].Pos("-");

        if(tseparator <= 1) continue;

        TEvent = TransListBox->Items->Strings[i].SubString(tlen, 1);

        if(MainU == TEvent)
        {
            if(TransListBox->Items->Strings[i].SubString(2, tseparator-2) == "") continue;

            t_num = StrToInt(TransListBox->Items->Strings[i].SubString(2, tseparator-2));

            if(t_num <= trans_count)
            {
                u_index++;

                for(int j=1; j<trans_count+1; j++)
                {
                    if(j == 1)
                    {
                        if(j == t_num)
                        {
                            UTuple = UTuple + "1";
                            U[j][u_index] = 1;
                        }
                    }
                    else
                    {
                        UTuple = UTuple + "0";
                        U[j][u_index] = 0;
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            if(j == t_num)
            {
                UTuple = UTuple + " 1";
                U[j][u_index] = 1;
            }
            else
            {
                UTuple = UTuple + " 0";
                U[j][u_index] = 0;
            }
        }
    }
    UListBox->Items->Add(UTuple);
    EListBox1->Items->Add(TEvent);
}
}
}

// Entera basıldığında Olay giriş alanı 1 deaktif edilir.
if(Key == VK_RETURN)
{
    U1AnilMemo->Enabled = false;
}

}
//-----

// U1AnilMemo alanına giriş yapıldığında aşağıdaki kontroller koşar.
void __fastcall TForm1::U1AnilMemoKeyPress(TObject *Sender, char &Key)
{
    // Sayı ve boşluk tuşlarından giriş kabul edilmez.
    if((Key == VK_SPACE)||((Key >= '0')&&(Key <= '9'))
        Key = 0x0;

    int len = U1AnilMemo->Text.Length();

    // Eğer son girilen değer ',' değil ise bir olay girildi demek ve hemen
    // arkasından ',' ya da enter ya da backspace tuşuna basılmalı.
    if((U1AnilMemo->Text.SubString(len,1) != ",")&&(len != 0))
    {
        if((Key != ',')&&(Key != VK_RETURN)&&(Key != VK_BACK))
            Key = 0x0;
    }

    // Eğer alan boş ise yada son olarak ',' girilmişse yeni bir virgül girişine izin verilmez.
    if(Key == ',')
    {
        if((U1AnilMemo->Text == "")||(U1AnilMemo->Text.SubString(len,1) == ","))
            Key = 0x0;
    }
}
//-----

```

- “Olay Giriş Alanı – 2” kullanılarak: “U1OzdeMemo” alanına ilişkin “U1OzdeMemoKeyDown” ve “U1OzdeMemoKeyPress” fonksiyonları tanımlıdır. U1OzdeMemoKeyPress fonksiyonunda giriş alanına sayı girilmesi ve space tuşuna basılması engellenir. Olayların tek tek ve aralarına virgöl konularak girilmesi sağlanır. İşlem bittiğinde ya ENTER tuşuna basılmalı veya virgöl konulmalıdır. U1OzdeMemoKeyDown fonksiyonunda, girilen her “e” olayının DGStringGrid’deki sütun indeksi “e_indeks” bulunur. Sistemin durum değiştirip değiştirmediği belirlenir, $DGStringGrid[e_indeks][X_o] = X_k \neq “ ”$ ise sistem durum değiştirmiş demektir. Sistemi X_k . duruma geçiren t-geçiş PNStringGrid tablosundan bulunur. Bu durumda u vektörü t geçişine karşılık düşen elemanı “1”, diğerleri “0” olacak şekilde düzenlenerek “U Vektör Alanı”na, “UListBox”, eklenir. Eğer sistem durum değiştirmiyorsa u vektörü tüm elemanları “0” olacak şekilde eklenir. Bir sonraki u çevrimi için ulaşılan yeni durum X_k , sistemin bulunduğu X_o durumuna atanır. Eğer sistem açmaz bir duruma ulaşırsa başlangıç durumuna döndürülür.

U1OzdeMemo alanına giriş tamamlandığında bu alanın yanındaki “ X_{k+1} Hesapla” tuşuna basılmalıdır. Bu tuşa basıldığında “OzdeXkButtonClick” fonksiyonu çalışır. Bu fonksiyonda UListBox’taki her u vektörü için $X_k[j] = X_o[j] + UA[j]$ değeri hesaplanır, eğer elde edilen durum vektörünün herhangi bir elemanı sıfırdan küçükse sistem bu u girişi için durum değiştirmez ve sistem durumunu korur ($X_k=X_o$). Aksi halde belirlenen yeni durum “ X_k ”, bir önceki durumdan “ X_o ”, farklı ise sistem durum değiştirmiş demektir. Yeni durum X_o ’a atanır ($X_o=X_k$). Sistem durum değiştirsin ya da değiştirmesin yapılan her hesaplama için X_k durum vektörü “ X_{k+1} Durumları” alanına eklenir. Eğer sistem açmaz bir duruma ulaşırsa başlangıç durumuna döndürülür.

```
// U1OzdeMemo alanına giriş yapılırsa aşağıdaki kontroller koşar.
// Olaylar tek tek ve aralarına virgöl konularak girilir.
void __fastcall TForm1::U1OzdeMemoKeyPress(TObject *Sender, char &Key)
{
    // U1OzdeMemo alanına sayı ya da boşluk girilemez.
    if((Key == VK_SPACE)||((Key >= '0')&&(Key <= '9'))
        Key = 0x0;

    int len = U1OzdeMemo->Text.Length();

    // Eğer son girilen değer ',' değil ise bir olay girildi demek ve hemen
    // arkasından ',' ya da enter ya da backspace tuşuna basılmalı.
    if((U1OzdeMemo->Text.SubString(len,1) != ",")&&(len != 0))
    {
        if((Key != ',')&&(Key != VK_RETURN)&&(Key != VK_BACK))
            Key = 0x0;
    }
}
```

```

    }
    // Eğer ekran boş ise ya da son olarak ',' girilmişse yeni bir virgül girişine izin verilmez.
    if(Key == ',')
    {
        if((U1OzdeMemo->Text == "")||(U1OzdeMemo->Text.SubString(len,1) == ","))
            Key = 0x0;
    }
}
//-----

// U1OzdeMemo alanına olaylar aralarına virgül konularak sırayla girilir.
// [1 0 .. 0] durumundan başlayarak oluşan olaylara göre gidilecek yeni
// durumlar belirlenir.
void __fastcall TForm1::U1OzdeMemoKeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    int tseparator,text_len,t_num,tlen;

    AnsiString UTuple,MainU,TEvent;

    UTuple = "";
    MainU = "";
    TEvent = "";

    t_num = 0;
    text_len = 0;
    tseparator = 0;
    tlen = 0;
    // Virgül ya da entera basılırsa
    if((Key == 188)||(Key == VK_RETURN))
    {
        if(DGStringGrid->Cells[1][0] == "")
        {
            Application->MessageBox("Lütfen Durum Geçiş Tablosunu
            oluşturunuz!", "Uyarı", MB_OK);
            U1OzdeMemo->Text = "";
            return;
        }

        // Durumların sayısı Durum Geçiş Matrisinin satır sayısının 1 eksiğine,
        // geçiş sayısı TransListbox'ın eleman sayısına eşittir.
        int place_count = DGStringGrid->RowCount - 1;
        int trans_count = TransListBox->Items->Count;

        if(AListBox->Items->Count == 0)
        {
            Application->MessageBox("Lütfen Önce PetriNet'e Geçiniz!", "Uyarı", MB_OK);
            U1OzdeMemo->Text = "";
            return;
        }

        // Girilen verinin uzunluğu hesaplanır.
        text_len = U1OzdeMemo->Text.Length();

        // U1OzdeMemo alanına olaylar tek tek ve aralarına virgül
        // konularak girilir, bu yüzden virgül ya da entera her basıldığında
        // U1OzdeMemo'daki verinin son harfi girilen olaya, MainU'ya, eşittir.
        MainU = U1OzdeMemo->Text.SubString(text_len,1);

        // Girilen olayın Durum Geçiş Matrisindeki sütun indeksi bulunur.
        int col_no = find_column_no(MainU,1);

        // u_index'i bir arttırılır.
        u_index = u_index + 1;

        // Durum Geçiş Matrisi'nden MainU olayı olduğunda hangi duruma
        // gidileceği bulunur. Eğer matristeki karşılığı boş ise sistem

```

```

// durum deęiřtirmiyor demektir.U matrisine [0 0 .. 0 ] atanır ve
// UListBox'a eklenir.
if (DGStringGrid->Cells[col_no][previous_state] == "" )
{
    for(int i=1;i<TransListBox->Items->Count+1; i++)
    {
        UTuple = UTuple + " 0" ;
        U[i][u_index] = 0;
    }

    UListBox->Items->Add(UTuple);
}
else
{
    // Eęer matristeki karřılıęı boş deęil ise hangi duruma (next_state) gidileceęi bulunur.
    int next_state = StrToInt(DGStringGrid->Cells[col_no][previous_state]);

    // Aktif olan t geęiři (previous_state'ten next_state'e geęiren)
    // PNStringGrid tablosundan bulunur ve U vektörüne çevrilir.

    for(int k=1;k<TransListBox->Items->Count+1;k++)
    {
        if(PNStringGrid->Cells[k][previous_state] == next_state)
        {
            for(int i=1;i<TransListBox->Items->Count+1; i++)
            {
                if (i == k)
                {
                    UTuple = UTuple + " 1" ;
                    U[i][u_index] = 1;
                }
                else
                {
                    UTuple = UTuple + " 0" ;
                    U[i][u_index] = 0;
                }
            }
            break;
        }
    }

    // Oluřan olaya iliřkin hesaplanan UTuple, UListBox'a eklenir.
    // Ve bir sonraki çevrim için previous_state güncellenir.
    UListBox->Items->Add(UTuple);
    previous_state = next_state;

    // Eęer ulařılan durum açmaz (deadlock) bir durum ise bařlangıç
    // durumuna geri dönülür.
    for (int i=0; i< 100; i++)
    {
        // final_states tablosu boş ise döngüden çıkılır.
        if (final_states[i] == 0)
            break;
        // Bařlangıç durumuna geri dönülür.
        if (next_state == final_states[i])
        {
            previous_state = 1;
            break;
        }
    }
}
}

// Enter tuřuna basılınca U1OzdeMemo alanına giriř yapılmasına izin verilmez.
if(Key == VK_RETURN)
{
    U1OzdeMemo->Enabled = false;
}

```

```

    }
}
//-----

// Alttaki "Xk+1 Hesapla" tuşuna basıldı.
// UListBox'daki tüm U'lar için  $X_{k+1} = X_k + uA$  hesaplanır,
// başlangıç durumu [1,0,0,...] olarak alınır. UListBox'a "Olay Giriş Alanı 2" ya da
// "U Giriş Alanı" kullanılarak giriş yapıldı ise bu tuşa basılmalıdır. "Olay Giriş Alanı 1"
// alanından giriş yapıldıysa üstteki "Xk+1 Hesapla" tuşuna basılmalıdır.
void __fastcall TForm1::OzdeXkButtonClick(TObject *Sender)
{
    AnsiString Xk_Row, Xo_Row;
    int UA[100],u[100],j, state;

    state = 0;
    XkListBox->Items->Clear();

    if(UListBox->Items->Count == 0)
    {
        Application->MessageBox("Lütfen U'yu giriniz!", "Uyarı", MB_OK);
        return;
    }

    // UA ve Xo sıfırlanır.
    for(int i=0;i<100;i++)
    {
        UA[i] = 0;
    }
    for(int i=0;i<100;i++)
    {
        Xo[i] = 0;
    }

    // Başlangıç durumu [1,0,0,...]
    Xo[1] = 1;

    // UListBox'daki tüm vektörler teker teker  $X_{k+1} = X_k + uA$  işlemine tabii
    // tutulur. u_index UListBox alanındaki U'ların toplam sayısıdır.
    for(int k=1;k<u_index+1;k++)
    {
        // UListBox alanındaki U'lar, lokal bir değişken olan u'ya atanır.
        for(int i=1;i<100;i++)
        {
            u[i] = U[i][k];
        }

        // Durumların sayısı Durum Geçiş Matrisinin satır sayısının bir eksiğine,
        // geçiş sayısı TransListBox'ın eleman sayısına eşittir.
        int place_count = DGStringGrid->RowCount - 1;
        int trans_count = TransListBox->Items->Count;

        // Matris çarpımında kullanılan UA, her u değişiminde sıfırlanır.
        for(int i=0;i<100;i++)
        {
            UA[i] = 0;
        }

        // uA çarpımı hesaplanır.
        for(int i=1;i<trans_count+1;i++)
        {
            for(int j=1;j<place_count+1;j++)
            {
                UA[j] = UA[j] + u[i] * a[i][j];
            }
        }
    }
}

```

```

//  $X_k+1 = X_k + uA$  işlemi sonucu hesaplanır. Eğer vektörün elemanlarından
// herhangi biri negatif ise sistem durum değiştirmez.
for(j=1;j<place_count+1;j++)
{
    Xk[j] = Xo[j] + UA[j];

    if(Xk[j] < 0)
        break;
}

// Eğer Xk negatif ise sistem durum değiştirmeyeceği için, bir önceki
// durum yani Xo, Xk'ya atanır.
if(j < place_count+1)
{
    for(int j=1;j<place_count+1;j++)
    {
        Xk[j] = Xo[j];
    }
}
// Sistemin şu anki durumu, bir sonraki u çevrimi için Xo'a atanır.
for(int j=1;j<place_count+1;j++)
{
    Xo[j] = Xk[j];
}

// Bulunan yeni durum, Xk vektörü, text olarak(Xk_Row) kaydedilir.
Xk_Row = "";
Xo_Row = "";
for(int i=1;i<place_count+1;i++)
{
    if ( Xk[i] == 1 )
        state = i;
    // Vektör elemanları aralarında boşluk olacak şekilde text olarak kaydedilir.
    if(i == 1)
        Xk_Row = IntToStr(Xk[i]);
    else
        Xk_Row = Xk_Row + " "+IntToStr(Xk[i]);
}

// Yeni durum XkListBox'a eklenir.
XkListBox->Items->Add(Xk_Row);

// Eğer gelinen durum açmaz(deadlock) durum ise başlangıç durumuna geri dönlür.
for(int index=0;index<100;index++)
{
    if ( state == final_states[index] )
    {
        for(j=1;j<place_count+1;j++)
        {
            // [1 0 .. 0] başlangıç durumu text olarak "Xo_Row"a kaydedilir.
            Xo[j] = 0;
            if(j == 1)
                Xo[j] = 1;
            // Vektör elemanları aralarında boşluk olacak şekilde text olarak kaydedilir.
            if(j == 1)
                Xo_Row = "1";
            else
                Xo_Row = Xo_Row + " "+"0";
        }
        // Başlangıç durumu XkListBox'a eklenir.
        XkListBox->Items->Add(Xo_Row);
        break;
    }
}
}
}

```

- “U Giriş Alanı” kullanılarak: “U2VektörMemo” alanına ilişkin “U2VektörMemoKeyDown”, “U2VektörMemoKeyPress” ve “U2VektörMemoEnter” fonksiyonları tanımlıdır. Bir olaya ilişkin u vektörü girilip ENTER tuşuna basılmalı ve vektör elemanları aralarına boşluk konularak girilmelidir. U2VektörMemoKeyPress fonksiyonunda sayı, space, back ve ENTER'dan başka bir tuşa basılmasına izin verilmez. U2VektörMemoEnter fonksiyonu her ENTER tuşuna basıldığında çalışır ve U2VektörMemo alanının bir sonraki yeni giriş için boşaltılmasını sağlar. U2VektörMemoKeyDown fonksiyonunda girilen u vektörlerinin boyutları kontrol edilir, t-geçiş sayısına eşit olmalıdır. Eksik girişler kabul edilmez, kullanıcı uyarılır. Fazla girişlerde ise kullanıcı uyarılır ancak vektörün t-geçiş sayısına kadar olan kısmı u girişi olarak kabul edilir. Kabul edilen girişler UlistBox’a eklenir.

U2VektörMemo alanına giriş tamamlandığında bu alanın yanındaki “ X_{k+1} Hesapla” tuşuna basılmalıdır. Bu tuşa basıldığında “OzdeXkButtonClick” fonksiyonu çalışır. Bu tuş aynı zamanda “Olay Giriş Alanı – 2” için kullanılan tuştur. Çalışma şekli 2. yöntemde anlatılmıştır.

```
// U Giriş Alanına her giriş yapıldığında enter tuşuna basılır ve U2VektörMemo
// alanı sıfırlanır.
void __fastcall TForm1::U2VektörMemoEnter(TObject *Sender)
{
    U2VektörMemo->Text = "";
}
//-----
// "U" alanına giriş yapıldığında aşağıdaki kontroller koşar.
void __fastcall TForm1::U2VektörMemoKeyPress(TObject *Sender, char &Key)
{
    // Sayı, boşluk, backspace ve enter'dan başka bir tuşa basılamaz.
    if(((Key < '0')||(Key > '9'))&&(Key != VK_SPACE)&&(Key != VK_BACK)&&(Key !=
VK_RETURN))
        Key = 0x0;

    int len = U2VektörMemo->Text.Length();

    // İlk giriş boşluk olamaz ve üst üste boşluk girilemez.
    if(Key == VK_SPACE)
    {
        if((U2VektörMemo->Text == "")||(U2VektörMemo->Text.SubString(len,1) == " "))
            Key = 0x0;
    }
}
//-----

// U Giriş Alanına vektör girilmesi durumu.
```

```

void __fastcall TForm1::U2VektorMemoKeyDown(TObject *Sender, WORD &Key,
TShiftState Shift)
{
int trans_count = 0;
int length,separator,main_len,j;
AnsiString U_Initial,MainU;

//Enter'a her basıldığında bir vektör girildi demektir.
if(Key == VK_RETURN)
{
    U2VektorMemo->Enabled = false;

    U2VektorMemo->Enabled = true;

    MainU = "";

    if(DGStringGrid->Cells[1][0] == "")
    {
        Application->MessageBox("Lütfen Durum Geçiş Tablosunu
        oluşturunuz!","Uyarı",MB_OK);
        return;
    }

    // Durumların sayısı Durum Geçiş Matrisinin satır sayısının 1 eksiğine,
    // geçiş sayısı TransListbox'ın eleman sayısına eşittir.
    int place_count = DGStringGrid->RowCount - 1;
    trans_count = TransListBox->Items->Count;

    main_len = 0;

    // U Giriş Alanına girilen vektörler U Vektör dizisi alanına eklenir ve bu
    // alandaki vektörlerin sayısını tutan u_index 1 arttırılır.
    UListBox->Items->Add(U2VektorMemo->Text);
    u_index++;

    U_Initial = U2VektorMemo->Text;

    length = U_Initial.Length();

    // Son girilen boşluk U_Initial'a dahil edilmez.
    if(U_Initial.SubString(length,1) == " ")
        U_Initial = U_Initial.SubString(1,length-1);

    // Girilen U_Initial ın boyutu t geçişleri sayısına eşit olmalıdır.
    // Eksik veya fazla olması durumları kontrol edilir.
    for(int i=1;i<100;i++)
    {
        if(U_Initial == "")
        {
            if(i < trans_count+1)
            {
                Application->MessageBox("Eksik veri girdiniz!","Uyarı",MB_OK);
                U2VektorMemo->Enabled = true;
                U2VektorMemo->Text = "";
                UListBox->Items->Clear();
                return;
            }
            if(i > trans_count+1)
            {
                Application->MessageBox("Fazla girilen verilen
                değerlendirilmemektedir!","Uyarı",MB_OK);
                return;
            }
        }
        return;
    }
}

```

```

length = U_Initial.Length();
separator = U_Initial.Pos(" ");

// U_Initial texti U Vektör dizisine ait U matrisine yazılır.
if(separator < 1)
{
    MainU = U_Initial.SubString(1,length);
    U_Initial = "";
}
else
{
    MainU = U_Initial.SubString(1,separator-1);

    main_len = MainU.Length();
    U_Initial = U_Initial.SubString(separator+1,length-separator);
}

if(MainU != "")
{
    if(i <= trans_count)
    {
        U[i][u_index] = StrToInt(MainU);
    }
}
}
}
}

```

“Tüm Alanları Temizle” tuşuna basıldığında “ClearBitBtnClick” fonksiyonu çalışır ve tüm alanların temizlenmesi sağlanır.

“U ve X_{k+1} Alanlarını Temizle” tuşuna basıldığında “ClearUXkBitBtnClick” fonksiyonu çalışır ve $x_{k+1} = x_k + u_k A$ hesaplanmasında kullanılan tüm alanlar temizlenir.

```

void __fastcall TForm1::ClearBitBtnClick(TObject *Sender)
{
    // 'Tüm Alanları Temizle' tuşuna basıldığında tüm alanlar silinir.
    Form1->clear_all();
}
//-----
void __fastcall TForm1::clear_all()
{
    int i,j;
    previous_state = 1;
    string_len = 0;
    LmListBox->Items->Clear();
    EventListBox->Items->Clear();
    LListBox->Items->Clear();
    StateListBox->Items->Clear();
    ALListBox->Items->Clear();
    XmListBox->Items->Clear();
    ULListBox->Items->Clear();
    XkListBox->Items->Clear();
}

```

```

TransListBox->Items->Clear();
EListBox->Items->Clear();
EListBox1->Items->Clear();
GirisMemo->Enabled = true;
U1AnilMemo->Enabled = true;
U2VektorMemo->Enabled = true;
U1OzdeMemo->Enabled = true;
GirisMemo->Text = "";
U1AnilMemo->Text = "";
U2VektorMemo->Text = "";
U1OzdeMemo->Text = "";
// Durum Geçiş Matrisi sıfırlanır.
for(i=0;i<Form1->DGStringGrid->ColCount; i++)
{
    for(j=0;j<Form1->DGStringGrid->RowCount; j++)
    {
        DGStringGrid->Cells[i][j] = "";
    }
}
Form1->DGStringGrid->RowCount = 2;
// Petri Net Geçiş Matrisi sıfırlanır.
for(i=0;i<Form1->PNStringGrid->ColCount; i++)
{
    for(j=0;j<Form1->PNStringGrid->RowCount; j++)
    {
        PNStringGrid->Cells[i][j] = "";
    }
}
Form1->PNStringGrid->RowCount = 2;
// UListBox a girilen U matrisinin indeksi
u_index = 0;
// Xk+1 i hesaplamakta kullanılan U ve a matrisleri sıfırlanır.
for(int i=0;i<100;i++)
{
    for(int k=0;k<100;k++)
    {
        U[i][k] = 0;
        a[i][k] = 0;
    }
}
// Xo ve Xk dizileri sıfırlanır.
for(int k=0;k<100;k++)
{
    Xo[k] = 0;
    Xk[k] = 0;
}

selected_col = 0;
selected_row = 0;
}
//-----

// "U ve Xk+1 Alanlarını Temizle" tusuna basıldı.
void __fastcall TForm1::ClearUXkBitBtnClick(TObject *Sender)
{
    // Global değişkenlere ilk değerleri atanır.
    previous_state = 1;
    string_len = 0;

    // Listboxlar sıfırlanır.
    UListBox->Items->Clear();
    XkListBox->Items->Clear();
    EListBox1->Items->Clear();

    // Olay ve U vektörü giriş alanlarına giriş yapılabilmesi sağlanır.
    U1AnilMemo->Enabled = true;
    U2VektorMemo->Enabled = true;

```

```

U1OzdeMemo->Enabled = true;

// Olay ve U vektörü giriş alanları sıfırlanır.
U1AnilMemo->Text = "";
U2VektorMemo->Text = "";
U1OzdeMemo->Text = "";

// UListBox a girilen U matrisinin indeksi sıfırlanır.
u_index = 0;

// U matrisi sıfırlanır.
for(int i=0;i<100;i++)
{
    for(int k=0;k<100;k++)
    {
        U[i][k] = 0;
    }
}

// Xo ve Xk dizileri sıfırlanır.
for(int k=0;k<100;k++)
{
    Xo[k] = 0;
    Xk[k] = 0;
}
}

```

“DG ile PLC Kodu Oluştur” tuşuna basıldığında “DGPLCBitBtnClick” fonksiyonu çalışır ve durum geçiş diyagramı yöntemini kullanan PLC SCL kodu üretilir. Bu kod programın çalıştırıldığı dizinin bir üst dizininde yaratılan PLC klasörü içinde “PLC_Code_DG.doc” olarak kaydedilir. Girilen her işaretli dil için ayrı bir kod üretilmektedir. İstenirse bu kod kopyalanarak “PLC Simatic Manager” ortamında derlenip çalıştırılabilir.

“A Matrisi ile PLC Kodu Oluştur” tuşuna basıldığında “APLCBitBtnClick” fonksiyonu çalışır ve A matrisi yöntemini kullanan PLC SCL kodu üretilir. Bu kod programın çalıştırıldığı dizinin bir üst dizininde yaratılan PLC klasörü içinde “PLC_Code_A.doc” olarak kaydedilir. Girilen her işaretli dil için ayrı bir kod üretilmektedir. İstenirse bu kod kopyalanarak “PLC Simatic Manager” ortamında derlenip çalıştırılabilir.

```

//----- PLC KODUNU HESAPLAMA KISMI -----

// "A Matrisi ile PLC Kodu Oluştur" ve "DG ile PLC Kodu Oluştur" tuşuna
// basılınca oluşan dosyaların kayıt edileceği PLC dizinini döndürür.
AnsiString PLCDirectory()
{
    AnsiString temp = ExtractFilePath(ParamStr(0));
    return ExpandFileName(temp + "..\\PLC\\");
}
//-----

// PLC kodunu Durum Geçiş Matrisinden hesaplama.

```

```

// "DG ile PLC Kodu Oluştur" tuşuna basıldı.
void __fastcall TForm1::DGPLCBitBtnClick(TObject *Sender)
{
    AnsiString    Dir;
    AnsiString    FileName,OldFileName;
    AnsiString
Message1,Message2,Message3,Message4,Message5,Message6,Message7,Message8,Message9,Message10,Messa
ge11;
    FILE          *fp;
    int            event_count=0;
    int            final_state_count=0;
    int            state_count=0;
    int            i,j;

    // PLC dizinindeki PLC_Code_DG.doc dosyası oluşturulup açılır.
    Dir = PLCDirectory();
    if (!DirectoryExists(Dir))
        ForceDirectories(Dir);

    FileName = Dir + "PLC_Code_DG.doc";

    if((fp = fopen(FileName.c_str(), "w")) == NULL)
    {
        return;
    }

    // PLC kodunun, girilen Lm(G)'ye bağlı olan kısımları hesaplanır.
    event_count = EventListBox->Items->Count;
    final_state_count = XmListBox->Items->Count-1;
    state_count = StateListBox->Items->Count;

    if(event_count == 0)
        return;

    Message10 = "";

    for( i=1;i<XmListBox->Items->Count-1;i++)
    {
        Message10 = Message10 + XmListBox->Items->Strings[i]+",";
    }
    Message10 = Message10 + XmListBox->Items->Strings[i];

    Message11 = "";

    for(i=1;i<DGStringGrid->ColCount; i++)
    {
        for(j=1;j<DGStringGrid->RowCount; j++)
        {
            if(DGStringGrid->Cells[i][j] == "")
            {
                if((i == DGStringGrid->ColCount-1)&&(j == DGStringGrid->RowCount-1))
                    Message11 = Message11 + "0";
                else
                    Message11 = Message11 + "0"+",";
            }
            else
            {
                if((i == DGStringGrid->ColCount-1)&&(j == DGStringGrid->RowCount-1))
                    Message11 = Message11 + DGStringGrid->Cells[i][j];
                else
                    Message11 = Message11 + DGStringGrid->Cells[i][j]+",";
            }
        }
    }

    Message1 = "// PLC giriş değerleri ve ON/OFF durumları \n";

```

```

Message2 = "I_STATECOUNTER:ARRAY[1.." + IntToStr(event_count) + ",1..2] OF BOOL :=
"+IntToStr(event_count) + "(0),"+IntToStr(event_count) + "(0); \n";
Message3 = "I_INITIALSTATE:ARRAY[1.." + IntToStr(event_count) + "] OF BOOL :=
"+IntToStr(event_count) + "(0); \n\n";
Message4 = "EVENTCOUNT:INT:="+IntToStr(event_count) + "; // Girilen olay sayısı \n";
Message5 = "FINALSTATECOUNT:INT:="+IntToStr(final_state_count) + "; // İşaretli durum sayısı \n\n";

Message6 = "// Borland kodundan aktarılan işaretli durumlar \n";
Message7 = "FINAL_STATES:ARRAY[1.." + IntToStr(final_state_count) + "] OF INT:= "+Message10 + ";
\n";
Message8 = "//Borland kodundan aktarılan Durum Geciş Matrisi \n";
Message9 = "DURUMGECISM:ARRAY[1.." + IntToStr(event_count) + ",1.." + IntToStr(state_count) + "] OF
INT:= "+Message11 + "; \n\n";

// PLC kodunun girilen Lm(G)'ye bağlı olmayan sabit kısımları PLC_Code_DG.doc'a eklenir.
fwrite(DGMemo1->Text.c_str(),DGMemo1->Text.Length(),1,fp);

// PLC kodunun, girilen Lm(G)'ye bağlı olan kısımları PLC_Code_DG.doc'a eklenir.
fwrite(Message1.c_str(),Message1.Length(),1,fp);
fwrite(Message2.c_str(),Message2.Length(),1,fp);
fwrite(Message3.c_str(),Message3.Length(),1,fp);
fwrite(Message4.c_str(),Message4.Length(),1,fp);
fwrite(Message5.c_str(),Message5.Length(),1,fp);
fwrite(Message6.c_str(),Message6.Length(),1,fp);
fwrite(Message7.c_str(),Message7.Length(),1,fp);
fwrite(Message8.c_str(),Message8.Length(),1,fp);
fwrite(Message9.c_str(),Message9.Length(),1,fp);

fwrite(DGMemo3->Text.c_str(),DGMemo3->Text.Length(),1,fp);

fclose(fp);
}
//-----

// PLC kodunu A Matrisinden hesaplama.
// "A Matrisi ile PLC Kodu Oluştur" tuşuna basıldı.
void __fastcall TForm1::APLCBitBtnClick(TObject *Sender)
{
    AnsiString Dir;
    AnsiString FileName,OldFileName,TIndex;
    AnsiString
Message1,Message2,Message3,Message4,Message5,Message6,Message7,Message8,Message9,Message10,

Message11,Message12,Message13,Message14,Message15,Message16,Message17,Message18,Message19,
Message20,

Message21,Message22,Message23,Message24,Message25,Message26,Message27,Message28,Message29,
Message30,

Message31,Message32,Message33;
    FILE *fp;
    int event_count=0;
    int final_state_count=0;
    int state_count=0;
    int trans_count=0;
    int i,j,tseparator;

    // PLC dizinindeki PLC_Code_A.doc dosyası oluşturulur ve açılır.
    Dir = PLCDirectory();
    if (!DirectoryExists(Dir))
        ForceDirectories(Dir);

    FileName = Dir + "PLC_Code_A.doc";

    if((fp = fopen(FileName.c_str(), "w")) == NULL)
    {
        return;
    }
}

```

```

// PLC kodunun, girilen Lm(G)'ye bağılı olan kısımları hesaplanır.
event_count = EventListBox->Items->Count;
final_state_count = XmListBox->Items->Count-1;
state_count = StateListBox->Items->Count;
trans_count = TransListBox->Items->Count;

if(event_count == 0)
    return;

if(AListBox->Items->Count == 0)
    return;

Message30 = "";

for( i=1;i<XmListBox->Items->Count-1;i++)
{
    Message30 = Message30 + XmListBox->Items->Strings[i]+",";
}
Message30 = Message30 + XmListBox->Items->Strings[i];

Message31 = "";

for(i=1;i<DGStringGrid->ColCount; i++)
{
    for(j=1;j<DGStringGrid->RowCount; j++)
    {
        if(DGStringGrid->Cells[i][j] == "")
        {
            if((i == DGStringGrid->ColCount-1)&&(j == DGStringGrid->RowCount-1))
                Message31 = Message31 + "0";
            else
                Message31 = Message31 + "0"+",";
        }
        else
        {
            if((i == DGStringGrid->ColCount-1)&&(j == DGStringGrid->RowCount-1))
                Message31 = Message31 + DGStringGrid->Cells[i][j];
            else
                Message31 = Message31 + DGStringGrid->Cells[i][j]+",";
        }
    }
}

Message32 = "";

for(i=0;i<TransListBox->Items->Count;i++)
{
    tseparator = TransListBox->Items->Strings[i].Pos("-");

    if(tseparator > 1)
        TIndex = TransListBox->Items->Strings[i].SubString(tseparator+1,1);

    int col_no = find_column_no(TIndex,1);

    if(i == TransListBox->Items->Count-1)
        Message32 = Message32 + IntToStr(col_no);
    else
        Message32 = Message32 + IntToStr(col_no)+",";
}

Message33 = "";

for(i=1;i<trans_count+1; i++)
{

```

```

for(j=1;j<state_count+1;j++)
{
    if((i == trans_count)&&(j == state_count))
        Message33 = Message33 + IntToStr(a[i][j]);
    else
        Message33 = Message33 + IntToStr(a[i][j])+", ";
}
}

Message1 = "FUNCTION_BLOCK FB1 \n\n";
Message2 = "VAR_TEMP \n";
Message3 = "I,J,M,K,L:INT; \n";
Message4 = "INDEX:INT; \n";
Message5 = "STATE_CHANGED:BOOL; \n";
Message6 = "FF_STATE:BOOL; // Deadlock olma durumu \n";
Message7 = "U1:ARRAY[1.."+IntToStr(trans_count)+"] OF INT; \n";
Message8 = "UA:ARRAY[1.."+IntToStr(state_count)+"] OF INT; \n";
Message9 = "END_VAR \n\n";
Message10 = "VAR \n";
Message11 = "XK:ARRAY[1.."+IntToStr(state_count)+"] OF INT:= "+IntToStr(state_count)+"(0); \n";
Message12 = "X0:ARRAY[1.."+IntToStr(state_count)+"] OF INT; \n\n";
Message13 = "// Borland kodundan aktarılan işaretli durumlar \n";
Message14 = "FINAL_STATES:ARRAY[1.."+IntToStr(final_state_count)+"] OF INT:= "+Message30+"; \n";
Message15 = "//Borland kodundan aktarılan geçişler \n";
Message16 = "TRANSITIONS:ARRAY[1.."+IntToStr(trans_count)+"] OF INT:= "+Message32+"; \n";
Message17 = "//Borland kodundan aktarılan geçiş sayısı \n";
Message18 = "TRANS_COUNT:INT := "+IntToStr(trans_count)+"; \n";
Message19 = "//Borland kodundan aktarılan Durum Geçiş Matrisi \n";
Message20 = "DURUMGEÇISM:ARRAY[1.."+IntToStr(event_count)+" ,1.."+IntToStr(state_count)+"] OF INT:= "+Message31+"; \n\n";
Message21 = "//Borland kodundan aktarılan Petri Net A Matrisi \n";
Message22 = "A:ARRAY[1.."+IntToStr(trans_count)+" ,1.."+IntToStr(state_count)+"] OF INT:= "+Message33+"; \n";
Message23 = "// PLC giriş değerleri ve ON/OFF durumları \n";
Message24 = "I_STATECOUNTER:ARRAY[1.."+IntToStr(event_count)+" ,1..2] OF BOOL := "+IntToStr(event_count)+"(0),"+IntToStr(event_count)+"(0); \n";
Message25 = "I_INITIALSTATE:ARRAY[1.."+IntToStr(event_count)+"] OF BOOL := "+IntToStr(event_count)+"(0); \n\n";
Message26 = "EVENTCOUNT:INT:="+IntToStr(event_count)+"; // Girilen olay sayısı \n";
Message27 = "FINALSTATECOUNT:INT:="+IntToStr(final_state_count)+"; // işaretli durum sayısı \n";
Message28 = "STATECOUNT:INT:="+IntToStr(state_count)+"; // durum sayısı\n\n";

// PLC kodunun, girilen Lm(G)'ye bağlı olan kısımları PLC_Code_A.doc'a eklenir.
fwrite(Message1.c_str(),Message1.Length(),1,fp);
fwrite(Message2.c_str(),Message2.Length(),1,fp);
fwrite(Message3.c_str(),Message3.Length(),1,fp);
fwrite(Message4.c_str(),Message4.Length(),1,fp);
fwrite(Message5.c_str(),Message5.Length(),1,fp);
fwrite(Message6.c_str(),Message6.Length(),1,fp);
fwrite(Message7.c_str(),Message7.Length(),1,fp);
fwrite(Message8.c_str(),Message8.Length(),1,fp);
fwrite(Message9.c_str(),Message9.Length(),1,fp);
fwrite(Message10.c_str(),Message10.Length(),1,fp);
fwrite(Message11.c_str(),Message11.Length(),1,fp);
fwrite(Message12.c_str(),Message12.Length(),1,fp);
fwrite(Message13.c_str(),Message13.Length(),1,fp);
fwrite(Message14.c_str(),Message14.Length(),1,fp);
fwrite(Message15.c_str(),Message15.Length(),1,fp);
fwrite(Message16.c_str(),Message16.Length(),1,fp);
fwrite(Message17.c_str(),Message17.Length(),1,fp);
fwrite(Message18.c_str(),Message18.Length(),1,fp);
fwrite(Message19.c_str(),Message19.Length(),1,fp);
fwrite(Message20.c_str(),Message20.Length(),1,fp);
fwrite(Message21.c_str(),Message21.Length(),1,fp);
fwrite(Message22.c_str(),Message22.Length(),1,fp);

```

```

fwrite(Message23.c_str(),Message23.Length(),1,fp);
fwrite(Message24.c_str(),Message24.Length(),1,fp);
fwrite(Message25.c_str(),Message25.Length(),1,fp);
fwrite(Message26.c_str(),Message26.Length(),1,fp);
fwrite(Message27.c_str(),Message27.Length(),1,fp);
fwrite(Message28.c_str(),Message28.Length(),1,fp);

// PLC kodunun, girilen Lm(G)'ye bağılı olmayan sabit kısımları PLC_Code_A.doc'a eklenir.
fwrite(APLCMemo->Text.c_str(),APLCMemo->Text.Length(),1,fp);

fclose(fp);
}

```

6.2 PLC SCL Programı

SCL programında PLC girişleri olaylar, çıkışları ise işaretli durumlar olarak kabul edilmiştir. Bir olayın oluşması için ona karşılık düşen giriş anahtarının iki kez durum değiştirmesi gerekmektedir. Geline yeni durum iki ayrı yöntemle belirlenir. Bunlar “durum geçiş diyagramı” ve “Petri ağı A matrisi” yöntemleridir. Çıkışlar 1’den başlayarak işaretli durumlara karşı düşecek şekilde indekslenmiştir.

6.2.1 Durum Geçiş Diyagramı Yöntemi

Olaylara karşılık düşen giriş anahtarlarının ilk değerleri I_INITIALSTATE dizisinde, değer değiştirme sayıları ise I_STATECOUNTER dizisinde tutulur; bu değer “2” olduğunda, giriş anahtarı iki kez değer değiştirmiş yani olay oluşmuş kabul edilir. Bu dizilerin boyutları olay sayısı kadardır.

Olay sayısı, işaretli durum sayısı, işaretli durumlar ve durum geçiş matrisi Borland C++ kodu tarafından girilen işaretli dile göre hesaplanır.

PLC kodunun ilk çevriminde giriş anahtarlarının ilk değerleri saklanır ve otomatın ilk durumu PREVIOUSSTATE değişkeninde “1” olarak tutulur. Her PLC çevriminde olay sayısı kadar giriş anahtarının değer değiştirip değiştirmediği kontrol edilir. Eğer herhangi bir anahtar iki kez değer değiştirmişse o anahtara karşılık düşen olay oluşmuş kabul edilir ve olay EVENTID değişkeninde tutulur.

Durum geçiş matrisinin “DURUMGECISM” olay sayısı kadar sütunu, durum sayısı kadar satırı vardır. DURUMGECISM[x,y]=z gösterilimi; PLC y durumunda iken x olayı oluştuğunda z durumuna geçildiğini ifade eder. Bu yöntemle yeni bir olay oluştuğunda (EVENTID>0 koşulu sağlandığında) gelinen yeni durum belirlenir ve

PREVIOUSSTATE’e atanır. PLC çıkışları sıfırlanır ve DEBUGMODE değişkeninin değerine göre çıkışlar belirlenir. Programda geline her yeni durum gözlemlenmek istenirse DEBUGMODE değişkeni TRUE yapılarak tüm durumlar kendi değerlerini gösterecek şekilde çıkışa verilir. DEBUGMODE değişkeninin varsayılan değeri FALSE’tur yani sadece işaretli durumlar, indekslenerek çıkışta gözlemlenir. Eğer geline durum açmaz bir durum ise yani durum geçiş matrisinde o duruma ilişkin satırlar “0” ise PLC ilk durumu “1”e döndürülür. Her çevrim sonunda oluşan olaya ait işlemler bittiği için EVENTID değişkeni sıfırlanır.

FUNCTION_BLOCK FB1

```
VAR_TEMP
I:INT;
J:INT;
FF_STATE : BOOL ; // Deadlock olma durumu
M,K,L:INT;
END_VAR

VAR
PREVIOUSSTATE:INT; // PLC nin bir önceki durumu
FIRSTCYCLE:BOOL;
DEBUGMODE:BOOL; // TRUE iken durumu çıkışa verir
// FALSE iken işaretli durumu çıkışa verir
EVENTID:INT:=0; // Gerçekleşen olay (PLC girişi ON/OFF veya OFF/ON olduğunda olay gerçekleşmiş kabul
// edilir.

// PLC giriş değerleri ve ON/OFF durumları
I_STATECOUNTER:ARRAY[1..2,1..2] OF BOOL := 2(0),2(0);
I_INITIALSTATE:ARRAY[1..2] OF BOOL := 2(0);
EVENTCOUNT:INT:=2; // Girilen olay sayısı
FINALSTATECOUNT:INT:=3; // İşaretli durum sayısı

// Borland kodundan aktarılan işaretli durumlar
FINAL_STATES:ARRAY[1..3] OF INT:= 2,4,6;
//Borland kodundan aktarılan Durum Geçiş Matrisi
DURUMGEÇISM:ARRAY[1..2,1..6] OF INT:= 2,3,4,0,6,0,4,5,0,0,1,0;

END_VAR

BEGIN
// İlk çevrimde PLC girişlerinin ilk değerleri saklanır.
// Başlangıç durumu Borland tarafında olduğu gibi 1 kabul edilir.
IF FIRSTCYCLE = FALSE THEN
M:=1;
FOR K:=0 TO 12 DO
FOR L:=0 TO 7 DO
I_INITIALSTATE[M]:=I[K,L];
M := M + 1;
IF M = EVENTCOUNT+1 THEN // Eğer olay sayısı kadar girişi güncellediysek
EXIT; // bizim için yeterli, exit ile çıkabiliriz.
END_IF;
END_FOR;
IF M = EVENTCOUNT+1 THEN
EXIT;
END_IF;
END_FOR;
PREVIOUSSTATE:=1; // PLC nin ilk durumu
FIRSTCYCLE := TRUE;
END_IF;
```

```

DEBUGMODE := FALSE; // Debugmode TRUE olduğu zaman çıkış olarak otomatın durumunu verir.
// FALSE iken sadece işaretli durumlar çıkışa verilir.

```

```

// PLC giriş değerlerinin ON/OFF veya OFF/ON olup olmadığı kontrol edilir.
// Kontrol sonucunda gerçekleşen olay tesbit edilir.
M:=1;

```

```

FOR K:=0 TO 12 DO
  FOR L:=0 TO 7 DO
    IF I[K,L] <> I_INITIALSTATE[M] THEN
      IF I_STATECOUNTER[M,1] = TRUE THEN
        I_STATECOUNTER[M,2] := TRUE; // Giriş ON/OFF veya OFF/ON oldu
      ELSE
        I_STATECOUNTER[M,1] := TRUE; // Giriş ON veya OFF oldu
      END_IF;

      I_INITIALSTATE[M]:=I[K,L]; // Giriş değeri güncellenir.
      IF (I_STATECOUNTER[M,2] = TRUE) THEN
        EVENTID:=M;
        I_STATECOUNTER[M,1] := FALSE;
        I_STATECOUNTER[M,2] := FALSE;
        EXIT;
      END_IF;
    END_IF;

    M := M + 1;
    IF M = EVENTCOUNT+1 THEN // Eğer olay sayısı kadar counterı güncellediysek
      EXIT; // bizim için yeterli, exit ile çıkabiliriz.
    END_IF;
  END_FOR;

  IF M = EVENTCOUNT+1 THEN // Eğer olay sayısı kadar counterı güncellediysek
    EXIT; // bizim için yeterli, exit ile çıkabiliriz.
  END_IF;
END_FOR;

```

```

// Gerçekleşen olay durum değişimine sebep oluyorsa PLC nin durumu güncellenir.
// Gelineen yeni durum işaretli durum ise PLC nin çıkışına verilir.

```

```

IF EVENTID <> 0 THEN
  IF DURUMGECISM[EVENTID,PREVIOUSSTATE]<> 0 THEN
    PREVIOUSSTATE:=DURUMGECISM[EVENTID,PREVIOUSSTATE];
    // PLC çıkışları sıfırlanır.
    FOR K:=0 TO 12 DO
      FOR L:=0 TO 7 DO
        Q[K,L]:=FALSE;
      END_FOR;
    END_FOR;

```

```

// DEBUGMODE TRUE ise çıkış olarak PLC nin durumu verilir.
// FALSE ise işaretli duruma ait indeks çıkış olarak verilir.

```

```

IF DEBUGMODE = TRUE THEN
  K := PREVIOUSSTATE DIV 8;
  L := PREVIOUSSTATE MOD 8;
  Q[K,L]:= TRUE;
ELSE
  FOR I:=1 TO FINALSTATECOUNT DO
    IF PREVIOUSSTATE = FINAL_STATES[I] THEN
      FF_STATE := TRUE;
      FOR J:=1 TO EVENTCOUNT DO
        IF DURUMGECISM[J,PREVIOUSSTATE]<>0 THEN
          FF_STATE := FALSE;
        END_IF;
      END_FOR;

      K := I DIV 8;
      L := I MOD 8;

```

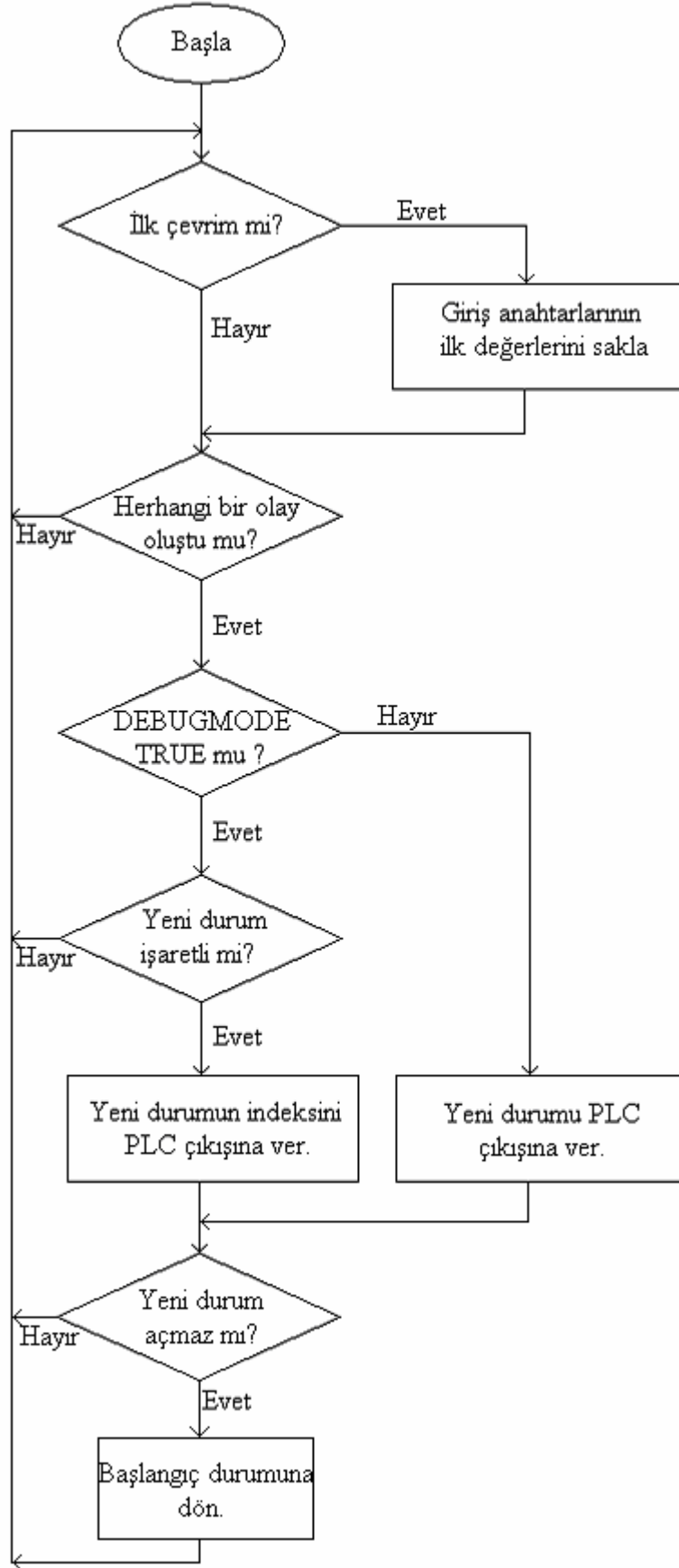
```
Q[K,L]:= TRUE;

// PLC nin durumu bir deadlock durumu ise ilk state'e döner.
IF FF_STATE = TRUE THEN
    PREVIOUSSTATE := 1;
END_IF;
END_IF;
END_FOR;
END_IF;

END_IF;

// Gerçekleşen olaya ait işlemler yapıldı. Bir sonraki çevrim için sıfırlanır.
EVENTID:=0;
END_IF;

END_FUNCTION_BLOCK
```



Şekil 6.3 : Durum Geçiş Diyagramı Yöntemi İşaret Akış Diyagramı

6.2.2 Petri Ağı A Matrisi Yöntemi

Olaylara karşılık düşen giriş anahtarlarının ilk değerleri I_INITIALSTATE dizisinde, değer değiştirme sayıları ise I_STATECOUNTER dizisinde tutulur; bu değer “2” olduğunda, giriş anahtarı iki kez değer değiştirmiş yani olay oluşmuş kabul edilir. Bu dizilerin boyutları olay sayısı kadardır.

Olay sayısı, durum sayısı, işaretli durum sayısı, t-geçiş sayısı, işaretli durumlar, durum geçiş matrisi, Petri ağı A matrisi ve t-geçişleri Borland C++ kodu tarafından girilen işaretli dile göre hesaplanır.

PLC kodunun ilk çevriminde giriş anahtarlarının ilk değerleri saklanır ve otomatın ilk durumu PREVIOUSSTATE değişkeninde “1” olarak tutulur, Xo durum dizisinde [1 0 0 .. 0] olacak şekilde tutulur. Xo dizisinin boyutu durum sayısı kadardır. Her PLC çevriminde olay sayısı kadar giriş anahtarının değer değiştirip değiştirmediği kontrol edilir. Eğer herhangi bir anahtar iki kez değer değiştirmişse o anahtara karşılık düşen olay oluşmuş kabul edilir ve olay EVENTID değişkeninde tutulur.

Durum geçiş matrisinin “DURUMGECISM” olay sayısı kadar sütunu, durum sayısı kadar satırı vardır. Petri ağı A matrisinin satırları t-geçişleri sayısı, sütunları ise durum sayısı kadardır.

$X_k = X_0 + u_k A$ denkleminde; X_0 PLC’nin bulunduğu durum, u_k oluşan olayın t-geçişine ait dizi, A Petri ağ matrisi ve X_k da geline yeni durumdur. Oluşan olaya ait her t-geçışı için u girişleri belirlenir. Her durumda bir t-geçışı aktiftir yani durum değişimine sebep olur. Geline yeni durum X_0 dizisine atanır.

PLC çıkışları sıfırlanır ve DEBUGMODE değişkeninin değerine göre çıkışlar belirlenir. Programda geline her yeni durum gözlemlenmek istenirse DEBUGMODE değişkeni TRUE yapılarak tüm durumlar kendi değerlerini gösterecek şekilde çıkışa verilir. DEBUGMODE değişkeninin varsayılan değeri FALSE’tur yani sadece işaretli durumlar, indekslenerek çıkışta gözlemlenir. Eğer geline durum açmaz bir durum ise yani durum geçiş matrisinde o duruma ilişkin satırlar “0” ise PLC ilk durumu “1”e döndürülür. Her çevrim sonunda oluşan olaya ait işlemler bittiği için EVENTID değişkeni sıfırlanır.

FUNCTION_BLOCK FB1

VAR TEMP
I,J,M,K,L:INT;
INDEX:INT;
STATE_CHANGED:BOOL;

```

FF_STATE:BOOL;    // Deadlock olma durumu
U1:ARRAY[1..7] OF INT;
UA:ARRAY[1..6] OF INT;
END_VAR

VAR
XK:ARRAY[1..6] OF INT:= 6(0);
X0:ARRAY[1..6] OF INT;

// Borland kodundan aktarılan işaretli durumlar
FINAL_STATES:ARRAY[1..3] OF INT:= 2,4,6;
//Borland kodundan aktarılan geçişler
TRANSITIONS:ARRAY[1..7] OF INT:= 1,1,1,2,1,2,2;
//Borland kodundan aktarılan geçiş sayısı
TRANS_COUNT:INT := 7;
//Borland kodundan aktarılan Durum Geçiş Matrisi
DURUMGECEISM:ARRAY[1..2,1..6] OF INT:= 2,3,4,0,6,0,4,5,0,0,1,0;

//Borland kodundan aktarılan Petri Net A Matrisi
A:ARRAY[1..7,1..6] OF INT:=-1,1,0,0,0,0,0,-1,1,0,0,0,0,0,-1,1,0,0,0,-1,0,0,1,0,0,0,-1,1,1,0,0,0,-1,0,-1,0,0,1,0,0;
// PLC giriş değerleri ve ON/OFF durumları
I_STATECOUNTER:ARRAY[1..2,1..2] OF BOOL := 2(0),2(0);
I_INITIALSTATE:ARRAY[1..2] OF BOOL := 2(0);

EVENTCOUNT:INT:=2;    // Girilen olay sayısı
FINALSTATECOUNT:INT:=3; // işaretli durum sayısı
STATECOUNT:INT:=6;    // durum sayısı

PREVIOUSSTATE:INT; // PLC nin bir önceki durumu

FIRSTCYCLE:BOOL;

DEBUGMODE:BOOL; // TRUE iken durumu çıkışa verir

// FALSE iken işaretli durumu çıkışa verir

EVENTID:INT:=0; // Gerçekleşen olay (PLC girişi ON/OFF veya OFF/ON olduğunda olay gerçekleşmiş kabul edilir.)

END_VAR

BEGIN

// İlk çevrimde PLC girişlerinin ilk değerleri saklanır.
// Başlangıç durumu Borland tarafında olduğu gibi 1 kabul edilir.
IF FIRSTCYCLE = FALSE THEN

    M:=1;

    FOR K:=0 TO 12 DO
        FOR L:=0 TO 7 DO
            I_INITIALSTATE[M]:=I[K,L];
            M := M + 1;

            IF M = EVENTCOUNT+1 THEN // Event sayısı kadar giriş yapıldıysa döngüden çıkılır.
                EXIT;
            END_IF;
        END_FOR;

        IF M = EVENTCOUNT+1 THEN // Event sayısı kadar giriş yapıldıysa döngüden çıkılır.
            EXIT;
        END_IF;
    END_FOR;

```

```

PREVIOUSSTATE:=1; // PLC nin ilk durumu
FIRSTCYCLE := TRUE;

// Başlangıç durum vektörü [1 0 ... 0]
FOR I:=1 TO STATECOUNT DO
    X0[I] := 0;
END_FOR;
X0[1] := 1;
END_IF;

DEBUGMODE := FALSE;

// PLC giriş değerlerinin ON/OFF veya OFF/ON olup olmadığı kontrol edilir.
// Kontrol sonucunda gerçekleşen olay tesbit edilir.

M:=1;
FOR K:=0 TO 12 DO
    FOR L:=0 TO 7 DO
        IF I[K,L] <> I_INITIALSTATE[M] THEN
            IF I_STATECOUNTER[M,1] = TRUE THEN
                I_STATECOUNTER[M,2] := TRUE; // Giriş ON/OFF veya OFF/ON oldu
            ELSE
                I_STATECOUNTER[M,1] := TRUE; // Giriş ON veya OFF oldu
            END_IF;

            I_INITIALSTATE[M]:=I[K,L]; // Giriş değeri güncellenir.
            IF (I_STATECOUNTER[M,2] = TRUE) THEN
                EVENTID:=M;
                I_STATECOUNTER[M,1] := FALSE;
                I_STATECOUNTER[M,2] := FALSE;
                EXIT;
            END_IF;
        END_IF;

        M := M + 1;
        IF M = EVENTCOUNT+1 THEN // Event sayısı kadar giriş yapıldıysa döngüden çıkılır.
            EXIT;
        END_IF;
    END_FOR;

    IF M = EVENTCOUNT+1 THEN // Event sayısı kadar giriş yapıldıysa döngüden çıkılır.
        EXIT;
    END_IF;
END_FOR;

//  $X_{k+1} = X_k + uA$  durum denklemi kullanılarak yeni durum belirlenir.
FOR I:=1 TO TRANS_COUNT DO
    U1[I] := 0;
END_FOR;

IF EVENTID <> 0 THEN

    FOR INDEX:=1 TO TRANS_COUNT DO
        STATE_CHANGED := TRUE ;
        IF TRANSITIONS[INDEX] = EVENTID THEN
            U1[INDEX] := 1;

            FOR I:=1 TO STATECOUNT DO
                UA[I] := 0;
            END_FOR;

            FOR I:=1 TO TRANS_COUNT DO
                FOR J:=1 TO STATECOUNT DO
                    UA[J] := UA[J] + U1[I] * A[I,J];
                END_FOR;
            END_FOR;
        END_IF;
    END_FOR;

```

```

FOR J:=1 TO STATECOUNT DO
    XK[J] := X0[J] + UA[J];
    IF XK[J] < 0 THEN
        EXIT;
    END_IF;
END_FOR;

IF J < STATECOUNT THEN
    STATE_CHANGED := FALSE ; // Gerçekleşen olay durum değişimine sebep olmuyor.
    FOR J :=1 TO STATECOUNT DO
        XK[J] := X0[J]; // Durum değiştirmedigi için bir önceki durum şimdiki duruma atanır
    END_FOR;
END_IF;

FOR J :=1 TO STATECOUNT DO
    X0[J] := XK[J]; // Bir sonraki döngüde kullanmak için X0 yeni durumla güncellenir.
END_FOR;

IF STATE_CHANGED THEN
    EXIT;
END_IF;

FOR I:=1 TO TRANS_COUNT DO
    U1[I] := 0;
END_FOR;

END_IF;
END_FOR;

IF STATE_CHANGED THEN
    // PLC çıkışları sıfırlanır.
    FOR K:=0 TO 12 DO
        FOR L:=0 TO 7 DO
            Q[K,L]:=FALSE;
        END_FOR;
    END_FOR;

    // Durum vektöründen yeni durum belirlenir.
    FOR J :=1 TO STATECOUNT DO
        IF XK[J] = 1 THEN
            PREVIOUSSTATE := J;
            EXIT;
        END_IF;
    END_FOR;

    // DEBUGMODE TRUE ise çıkış olarak PLC nin durumu verilir.
    // FALSE ise işaretli duruma ait indeks çıkış olarak verilir.
    IF DEBUGMODE = TRUE THEN
        K := PREVIOUSSTATE DIV 8;
        L := PREVIOUSSTATE MOD 8;
        Q[K,L]:= TRUE;
    ELSE
        FOR I:=1 TO FINALSTATECOUNT DO
            IF FINAL_STATES[I]=0 THEN
                EXIT;
            END_IF;

            IF PREVIOUSSTATE = FINAL_STATES[I] THEN
                FF_STATE := TRUE;
                FOR J:=1 TO EVENTCOUNT DO
                    IF DURUMGECISM[J,PREVIOUSSTATE]<>0 THEN
                        FF_STATE := FALSE;
                    END_IF;
                END_FOR;
            END_IF;
        END_FOR;
    END_IF;
END_FOR;

```

```
K := I DIV 8;
L := I MOD 8;
Q[K,L]:= TRUE;

// PLC nin durumu bir deadlock durumu ise ilk duruma döner.
IF FF_STATE = TRUE THEN
  PREVIOUSSTATE := 1;
  FOR I:=1 TO STATECOUNT DO
    X0[I] := 0;
  END_FOR;
  X0[1] := 1;
END_IF;

END_IF;
END_FOR;
END_IF;
END_IF;

// Gerçekleşen olaya ait işlemler yapıldı. Bir sonraki çevrim için sıfırlanır.
EVENTID:=0;
END_IF;

END_FUNCTION_BLOCK
```

KAYNAKLAR

Cassandras, Christos G. ve Lafortune, Stephane, 1999. Introduction To Discrete Event Systems, Kluwer Academic Publishers, Boston/Dordrecht/London.

Cassandras, Christos G., 1993. Discrete Event Systems: Modeling and Performance Analysis, Irwin Publ.

Yanik Memik, 2003. Borland C++ Builder, Seckin Yayıncılık, Ankara.

ÖZGEÇMİŞ

Özde Tiryaki 1977 yılında Manisa’da doğdu. Orta öğrenimini 1995 yılında Sekine Evren Anadolu Lisesi’nde tamamladı. 1999 yılında İstanbul Teknik Üniversitesi’nde Kontrol ve Bilgisayar Mühendisi diplomasını aldı. 2000 yılında telekomünikasyon alanında faaliyet gösteren Nortel Netas firmasında işe başladı. Halen bu firmada yazılım geliştirme mühendisi olarak çalışmaktadır.