İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİÇİMSEL DİLLERDEN ENDÜSTRİYEL İŞLEMCİLERE OTOMATİK KOD ÜRETME: PETRİ AĞ YAKLAŞIMI

YÜKSEK LİSANS TEZİ

Müh. Anıl ŞAHİN

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

OCAK 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİÇİMSEL DİLLERDEN ENDÜSTRİYEL İŞLEMCİLERE OTOMATİK KOD ÜRETME: PETRİ AĞ YAKLAŞIMI

YÜKSEK LİSANS TEZİ

Müh. Anıl ŞAHİN

504991068

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 Özde Tiryaki'ye ve aileme teşekkürü bir borç bilirim.

Ocak 2007 Anıl Şahin

İÇİNDEKİLER

KISALTMALAR

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

ŞEKİL LİSTESİ

Sayfa No

SEMBOL LİSTESİ

G	**:** Otomat
N	**:** Petri ağı
X	**:** Durum kümesi
E	**:** Olay kümesi
f	**:** Durum geçiş fonksiyonu
Γ	**:** Aktif olay kümesi
$\mathbf{x_0}$	**:** Başlangıç durumu
$\mathbf{X_m}$	**:** İşaretli durumlar
P	**:** Yer kümesi
$\mathbf{p_1,..,p_n}$	**:** n adet yer
T	**:** Geçiş kümesi
$\mathbf{t_1,..,t_m}$	**:** m adet geçiş
A	**:** Petri ağ matrisi
w	**:** Ark ağırlık fonksiyonu
l	**:** Geçişlerin etiketlenme fonksiyonu
u	**:** Ateşleme vektörü
I	**:** Giriş fonksiyonu
O	**:** Çıkış fonksiyonu
R	**:** Regüler dil
L	**:** Üretilen dil
$\mathbf{L_m}$	**:** İşaretli dil
ε	**:** Boş kelime

BİÇİMSEL DİLLERDEN ENDÜSTRİYEL İŞLEMCİLERE OTOMATİK KOD ÜRETME: PETRİ AĞ 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 sistemin Petri ağ modeli elde edilmiş ve bu model temel alınarak kullanılan işlemciye uygun kod üreten yazılım gerçeklenmiştir. 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: PETRI NET 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, petri net model of system is identified from state transition matrix and a software which generates automatic code from this model is developped in a language appropriate for the processor. This software is developped in Borland C++ Builder due to its visual properties. For industrial processor, SIMATIC-300 is choosen 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 sensör 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 kontrolsuz 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 kontrolu 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 metodların 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 uğraşılmaktadır. 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 kismi 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, kontrolu 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 ya da çıkış değerlerine bağlı değildir. Dinamik sistem de ise sistemin çıkışı, y(t), geçmişteki giriş ya da 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, $x(t) = [\, x_1, x_2, .. , x_n]^T$ 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,

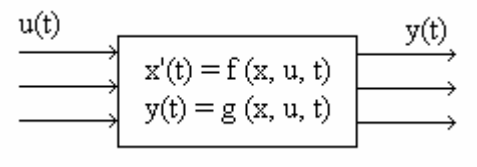
$$x'(t) = f(x(t), u(t), t) \qquad \textbf{(1.1)}$$

durum uzayı modeli ise durum ve çıkış denklemleri ile tanımlanır.

$$x'(t) = f(x(t), u(t), t) \quad x(t_0) = x_0 \text{(başlangıç koşulu)} \qquad \textbf{(1.2)}$$

$$y(t) = g(x(t), u(t), t) \qquad \textbf{(1.3)}$$

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(.) hem de g(.) fonksiyonlarının lineer olmasıdır. Lineer durumda (1.2) ve (1.3) denklemleri,

$$x'(t) = A(t)x(t) + B(t)u(t) \qquad \textbf{(1.4)}$$

$$y(t) = C(t)x(t) + D(t)u(t) \qquad \textbf{(1.5)}$$

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 ifadelere 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.

$$u(t) = \gamma(r(t), x(t), t) \qquad \textbf{(1.6)}$$

1.5 Ayrık 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 ayrık anlarında tanımlanmış olduğu varsayılsın. Bunun sonucu olarak, ayrık zaman sistemleri elde edilir. Ayrık zaman sistemlerini incelemenin nedenleri olarak; dijital bilgisayarların çalışma prensibi, diferansiyel denklemlerin nümerik çözümleri, dijital kontrol teknikleri ve dijital kontrolörler, bazı sistemlerin doğaları gereği ayrık zamanlı olmaları; "ekonomik sistemler" gösterilebilir.

Ayrık olay sistemlerinin durum uzayı, {0,1,2, .. } gibi ayrık bir kümeden oluşur ve durum geçişleri sadece zamanın ayrık 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 "Rastgele Yürüyüş":

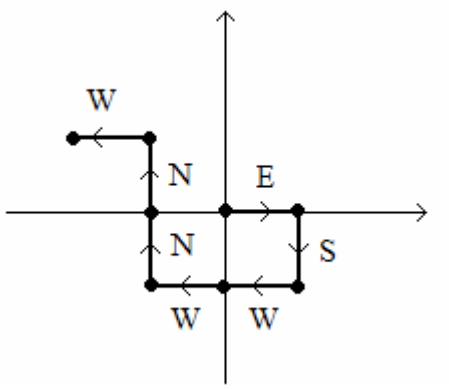
İki boyutlu bir alanda yapılan rastgele yürüyüş için, herhangi bir anda dört yöne (kuzey, güney, doğu, batı) doğru birim mesafe kadar hareket edilebildiği ve yönün rastgele seçildiği varsayımı yapılır. Sistemin durumu (x_1, x_2) konumlarıdır. x_1 ve x_2 sadece tam sayı değerleri alabilir. Yani durum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

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

Bu durumda, doğal bir olay kümesi;

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

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 oluştuğu varsayılırsa Şekil 1.2'deki durum yörüngesi (sample path) elde edilir.



Şekil 1.2 Rastgele 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 değişirler. Bu nedenle bu sistemlere "zaman denetimli" sistemler denir. Bu durumda, zaman değişkeni ($t \in R$ veya $k \in I$ dır.) bağımsız değişken olarak ortaya çıkar ve giriş, durum ve çıkış fonksiyonlarının argümanını oluşturur.

Ayrık durum sistemlerinde durumlar sadece belirli zaman noktalarında ani geçişler şeklinde değişir. 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 değişikliğine 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 değiştirir 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 süreçlerin 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 değişir. 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.

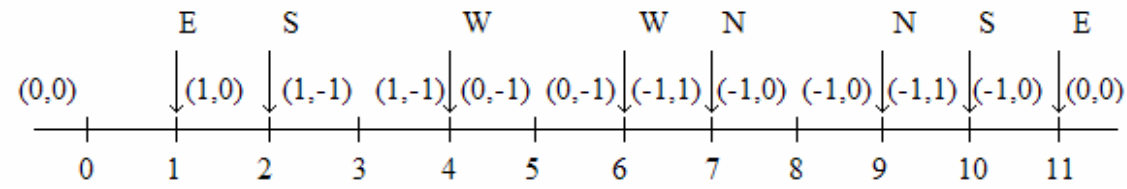
Olay denetimli sistemlerdeki durum geçişlerinin en iyi ve bilindik bir örneği bilgisayardaki "kesme" kavramıdır. Bir bilgisayarda bir çok işlem bir saat işareti ile senkronize edildiği, yani zaman denetimli olduğu halde, işletim sistemi zamanın herhangi bir anında oluşan çağrılara cevap verebilecek şekilde 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 Rastgele Yürüyüş":**

Şekil1.2'deki rastgele 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 rastgele yürüyüşün alternatif bir biçimi daha vardır, burada parçacığın hareketinin kontrolu 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 Rastgele Yürüyüş

Örnek 1.2'de iki olayın tam olarak aynı anda oluşamayacağı varsayılmıştır. Eğer böyle bir durum olursa 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 varsayılsın. 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 ugulama vardır.

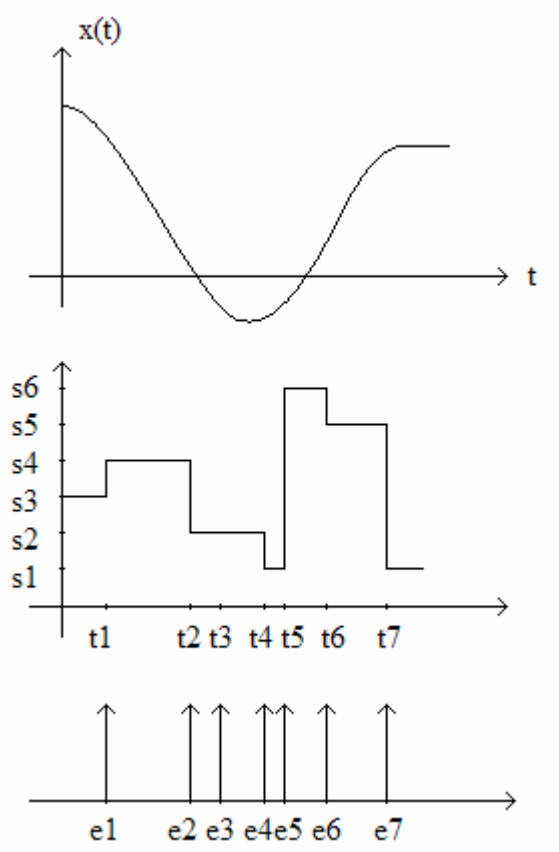
Bir makinanın durumu {On, Off} ya da {Meşgul, Boşta, Arızalı} olabilirken, bir programı çalıştıran bir bilgisayar şu üç durum içerisinde değerlendirilebilir {Giriş bekliyor, Çalışıyor, Bozuk}. Oyunların çoğu bir ayrık durum uzayına sahip olarak modellenebilir. Satrançta, örneğin her mümkün konfigürasyon 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 1.4 de gösterildiği gibi her sistem sınıfına ilişkin tipik yörüngelerinin karşılaştırılması ile açığa çıkar.

- CVDS de durum uzayı x Є R reel sayılar kümesidir ve x(t) Є 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 CVDS ve DES Grafikleri

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 hem de ayrık zamanlı olarak modellenebilirler. 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 çalışabilinmesi için kullanılan üç seviyeli soyutlamadır. Bu soyutlamalardan hangisinin seçileceği, analizin amacına bağlıdır. Her üç seviye de 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 '$e_1, e_2, ..., e_n$' 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 durumlardır.

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 gerçeklenmiştir. 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.

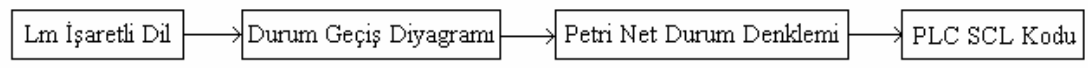
Bu çalışmada üretilen programın işaret akış diyagramı Şekil 1.5'de gösterildiği gibidir. Öncelikle, girilen bir işaretli dilin otomatı, "G", belirlenir.

$$G = \{X, E, f, \Gamma, x_0, X_m\} \tag{1.9}$$

Daha sonra G otomatının durum geçiş diyagramından "N" Petri ağına geçilir. Durum denklemleri ($X_{k+1} = X_k + u_k A$) kullanılarak oluşan olaylara karşı düşen yeni durumlar hesaplanır.

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

Hem durum geçiş 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. PETRİ AĞLARI

2.1 Giriş

Petri ağları, zamanlanmamış DES modellenmesi için otomatlara alternatif bir yöntem oluşturur. Petri ağ modelleri 1960 yılında C.A.Petri tarafından geliştirilmiştir. Petri ağları, otomatlarla DES'in geçiş fonksiyonlarını temsil etme anlamında ilişkilendirilirler. Otomattaki gibi, bir Petri ağı belirli kurallara bağlı olarak olaylarla yönetilen bir cihazdır. Petri ağlarının özelliklerinden biri hangi olayın mümkün olabileceğinin koşullara bağlı olmasıdır. Bu özellik, işlemleri karmaşık kontrol şemalarına bağlı olan çok genel DES'lerin gösteriliminin elde edilmesine imkan sağlar. Bu model grafik olarak göstermeye de uygun bir yapıdır. Ancak küçük sistemler için bu yapılabilir. Elde edilen grafiğe "Petri Ağ Grafı (Petri Net Graph)" denir. Petri ağ grafı, sistemle ilgili bir çok yapısal özelliği kapsar ve sezgiseldir. Her otomat bir Petri ağı ile gösterilebilir ancak her petri ağı bir otomat ile gösterilemez. Bu nedenle Petri ağları "R" regüler dillerden daha geniş bir dil sınıfını ifade edebilirler. Petri ağının başka bir avantajı da analiz teknikleri açısından daha güçlü olmasıdır. Bu teknikler sadece zamanlanmamış Petri ağlarını değil zamanlı Petri ağlarını da kapsar. Özellikle "max-plus algebra" olarak bilinen, zamanlı Petri ağlarının bir sınıfı için geliştirilmiş bir kuram vardır. Belirtilmesi gereken bir diğer konu ise PLC için yaygın olarak kullanılan "Grafcet" programlama dilinin Petri ağlarından esinlenilerek geliştirilmiş olmasıdır.

2.2 Petri Ağlarının Temelleri

Bir Petri ağının tanımı iki adımda yapılır. Birinci adım, otomatın durum geçiş diyagramına benzer olan, Petri ağ yapısı olarak da adlandırılan Petri ağ grafıdır. (Bölüm 2.2.1). İkinci adım ise tam Petri ağ modelinin oluşturulması için başlangıç durumu, işaretli durumlar, geçiş fonksiyonları, ilgili dinamikler ve üreten/belirleyen dillerin eklenmesidir. (Bölüm 2.2.2 - Bölüm 2.2.4).

2.2.1 Petri Ağ Notasyonları ve Tanımları

Petri ağlarında "olaylar" "geçiş"lere bağlanmıştır. Bir geçişin gerçekleşmesi bir çok koşulun sağlanmasını gerektirir. Bu koşullar ile ilgili bilgi "yer"lerde içerilir. Bazı böyle yerlere "geçişlerin girişleri" olarak bakılır. Diğer yerler ise "geçişlerin çıkışları" olacaktır. Geçişler, yerler ve aralarındaki ilişkiler Petri ağ grafının temel

elemanlarını oluşturur. Bir Petri ağı iki tip düğüme sahiptir; bunlar geçişler ve yerlerdir ve "ark (arc)"lar bunları birbirine bağlar. Arklar aynı tip iki düğümü birbirine bağlayamazlar. Geçiş düğümünü yer düğümüne veya yer düğümünü geçiş düğümüne bağlayabilirler. Bu özelliğe "iki taraflı, iki kısımlı olma (bipartite)" denir.

Tanım 2.1 "Petri Ağ Grafı (Petri Ağ Yapısı)":

Bir Petri ağ grafı (P, T, A, w) ağırlıklı (weighted) iki taraflı graftır.

"P", sonlu sayıda bir küme ve "yer"ler denir. (Grafın bir tip düğümüdür).

"T", sonlu sayıda bir küme ve "geçiş'ler denir. (Grafın diğer tip düğümüdür).

$A \subseteq (P \times T) \cup (T \times P)$, Graftaki yerlerden geçişlere, geçişlerden yerlere bağlantıyı sağlayan arkların kümesidir.

$w : A \to \{1, 2, 3, ...\}$, arkların ağırlık fonksiyonudur.

(P, T, A, w) bir izole yer ve geçişin olmadığı varsayılır. Yani bütünlük olacaktır.

Yer kümesi $P = \{p_1, p_2, ..., p_n\}$ ve geçiş kümesi $T = \{t_1, t_2, ..., t_m\}$ şeklinde gösterilebilir. Bunlar sayılabilir ve sonlu kümelerdir. Bir ark (p_i, t_j) veya (t_j, p_i) formundadır ve arkın ağırlığı pozitif bir sayıdır.

(p_i, t_j), $(t_j, p_i) \in A$ dır.

Bir Petri ağ grafının otomatın durum geçiş diyagramından daha karmaşık olduğu görülür. Öncelikle durum geçiş diyagramındaki düğümler X kümesindeki durumlardır. Petri ağ grafında ise düğümler P kümesindeki yerler veya T kümesindeki geçişlerdir. Durum geçiş diyagramında durum geçişine sebep olan her olaya ilişkin bir ark bulunurken Petri ağ grafında iki düğümü bağlayan çoklu arklara izin verilir ya da benzer olarak arkların sayısını temsil eden her arka bir ağırlık atanır. Bu nedenle bu yapı "çoklu graf (multigraph)" yapısı olarak adlandırılır.

Petri ağ grafında, $I(t_j)$ bir t_j geçişinin giriş yerlerinin kümesi, $O(t_j)$ bir t_j geçişinin çıkış yerlerinin kümesi olarak tanımlamak uygun olur.

$$I(t_j) = \{p_i \in P : (p_i, t_j) \in A\} \quad \textbf{(2.1)}$$

$$O(t_j) = \{p_i \in P : (t_j, p_i) \in A\} \quad \textbf{(2.2)}$$

(2.1) denklemi t_j'ye giriş olan p_i'leri, (2.2) denklemi t_j'ye çıkış olan p_i'leri gösterir.

Benzer notasyon giriş ve çıkış geçişleri olarak da tanımlanabilir.

$$I(p_i) = \{t_j \in T : (t_j, p_i) \in A\} \quad \textbf{(2.3)}$$

$$O(p_i) = \{t_j \in T : (p_i, t_j) \in A\} \quad \textbf{(2.4)}$$

(2.3) denklemi p_i'ye gelen t_j geçişler kümesini, (2.4) denklemi p_i'den giden t_j geçişler kümesini gösterir.

Petri ağ grafı çizilirken iki tip düğüm olan yerleri ve geçişleri birbirinden ayırmak gerekir. Bu nedenle yerleri göstermek için daire, geçişleri göstermek için bar kullanılır. Yerleri ve geçişleri bağlayan arklar A ark kümesinin elemanlarıdır. p_i yerinden t_j geçişine yönlendirilen ark $p_i \in I(t_j)$'dir.

$$w(p_i, t_j) = k \quad \textbf{(2.5)}$$

denklemi "p_i'den t_j'ye k adet ark var" ya da "p_i'den t_j'ye ağırlığı k olan bir ark var" anlamındadır.

Benzer olarak "t_j geçişinden p_i yerine k adet ark var" ifadesi $p_i \in O(t_j)$ ve

$$w(t_j, p_i) = k \quad \textbf{(2.6)}$$

şeklinde gösterilir. Genellikle bir grafta çoklu arklara ağırlıklar sunulur. Bununla beraber büyük ağırlıklar içeren bir Petri ağında arka ağırlık yazmak daha uygun bir gösterimdir. Eğer bir Petri ağında arka ağırlık yazılmamışsa ağırlık 1 kabul edilir. Son olarak da ağırlık fonksiyonunun tanım ve değer bölgesi genişletilerek aşağıdaki denklemler yazılabilir.

$$p_i \notin I(t_j) \text{ ise } w(p_i, t_j) = 0 \quad \textbf{(2.7)}$$

$$p_i \notin O(t_j) \text{ ise } w(t_j, p_i) = 0 \quad \textbf{(2.8)}$$

Örnek 2.1:

Basit bir Petri ağ grafı şöyle tanımlansın.

$P = \{p_1, p_2\}$, $T = \{t_1\}$, $A = \{(p_1, t_1), (t_1, p_2)\}$

$w(p_1, t_1) = 2$, $w(t_1, p_2) = 1$

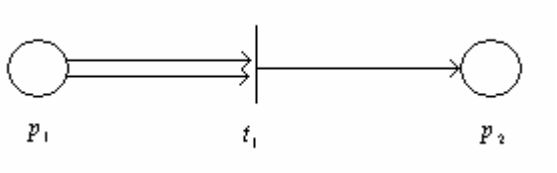
Bu durumda,

$I(t_1) = \{p_1\}$, $O(t_1) = \{p_2\}$

$I(p_1) = \{\phi\}$, $O(p_1) = \{t_1\}$

$I(p_2) = \{t_1\}$, $O(p_2) = \{\phi\}$

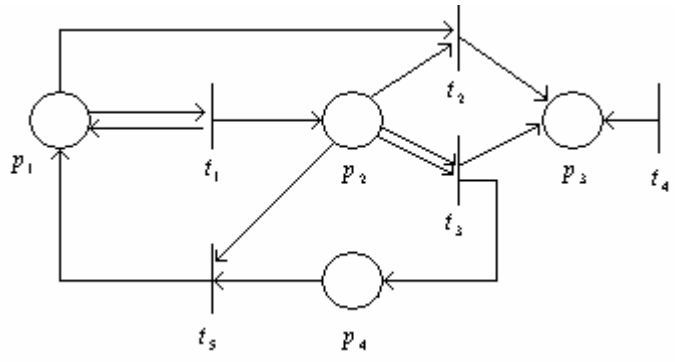
olacaktır. $w(p_1, t_1) = 2$ olması p_1 yerinden t_1 geçişine 2 ark olduğunu gösterir. Örneğe ilişkin Petri ağ grafı Şekil 2.1'de verilmiştir.



Şekil 2.1 : Örnek 2.1 Petri Ağ Grafı

Örnek 2.2:

Örnek 2.1'de tanımdan Petri ağ grafına geçilmiştir. Bu örnekte de Petri ağ grafından formal tanım oluşturulacaktır.



Şekil 2.2 : Örnek 2.2 Petri Ağ Grafı

$P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2, t_3, t_4, t_5\}$

$A = \{(p_1, t_1), (p_1, t_2), (p_2, t_2), (p_2, t_3), (p_2, t_5), (p_4, t_5), (t_1, p_1),$

$(t_1, p_2), (t_2, p_3), (t_3, p_3), (t_3, p_4), (t_4, p_3), (t_5, p_1)\}$

$w(p_1, t_1) = 1$ $w(p_1, t_2) = 1$ $w(p_2, t_2) = 1$ $w(p_2, t_3) = 2$

$w(p_2, t_5) = 1$ $w(p_4, t_5) = 1$ $w(t_1, p_1) = 1$ $w(t_1, p_2) = 1$

$w(t_2, p_3) = 1$ $w(t_3, p_3) = 1$ $w(t_3, p_4) = 1$ $w(t_4, p_3) = 1$

$w(t_5, p_1) = 1$

t_4 geçişi bir giriş yerine sahip değildir. Eğer geçişler olaylar ve yerler durumlar olarak düşünülürse bu t_4'e ilişkin olayın oluşunun bir koşula bağlanmadığı anlamına gelir. Tersine t_2 geçişi hem p_1 hem de p_2 koşullarına bağlıdır.

2.2.2 Petri Ağının İşaretlenmesi ve Durum Uzayları

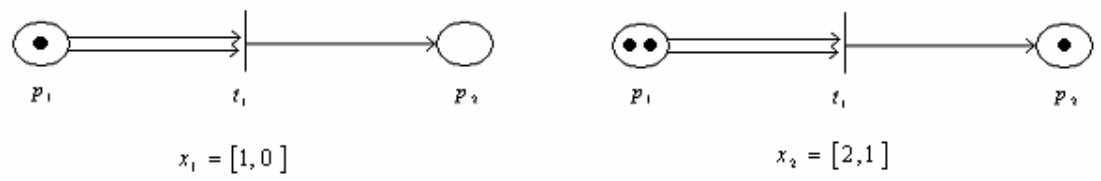
Geçişler bilindiği gibi otomatın durum geçişlerine karşı düşmelidirler. Yani bir olay olduğunda (olay izinli ise) DES durum değişikliği yapmalıdır. Şimdiye kadar durumlar ele alınmadı. Bir DES olayla sürülen yani durum değiştiren bir sistem olmak zorundadır. Durumları ve durum geçişlerini Petri ağ grafı üzeinde göstermek için "jeton"lar kullanılır. Jetonların yerlerdeki dağılımına "işaretleme (marking)" denir. Formal olarak bir (P, T, A, w) Petri ağının bir "x" işaretlemesi P(yer)'den N doğal sayılara bir fonksiyondur. $x : P \to N = \{0, 1, 2, ...\}$. Böylece x bir satır vektörü ile gösterilir. n Petri ağındaki yer sayısı olmak üzere $x = [x(p_1), x(p_2), ..., x(p_n)]$'dir. Bu vektörün i. elemanı $x(p_i)$, p_i yerinde bulunan jeton sayısıdır. Petri ağ graflarında jetonlar siyah noktalar ile gösterilir.

Tanım 2.2 "İşaretli Petri Ağı":

Bir işaretli Petri ağı (P, T, A, w, x) ile gösterilir. Burada (P, T, A, w) Petri ağ grafı ve x ise yerler kümesi P'nin bir işaretlemesidir. $x = [x(p_1), x(p_2), ..., x(p_n)] \in N$.

Örnek 2.3:

Şekil 2.1'deki Petri ağının olası iki işaretlemesi Şekil 2.3'te verilmiştir.



Şekil 2.3 : Şekil 2.1 Grafının İki İşaretlemesi

İşaretli Petri ağı yerine Petri ağı ifadesi kullanılır. Çünkü sistemin modeli durumları içermelidir. Böylece işaretlerin değişimi aslında durumlar olarak ortaya çıkacaktır. Jetonların yerlerdeki dağılımı keyfidir, herhangi bir sınırlama yoktur. Bu nedenle de genel olarak durum uzayı ∞'a gidebilir. Böylece n yere sahip bir Petri ağının X durum uzayı n boyutlu bir vektör uzayında tanımlanır ve elemanları negatif olmayan tamsayılardır, yani $X = N^n$. Petri ağ literatüründe "işaretleme", "durum" teriminden daha yaygındır. Durum kelimesi sistem dinamiklerini ifade ettiği için daha anlamlıdır. Durum kelimesinin kullanılması, otomatlardaki "işaretli durum (marked state)" (daha sonra Petri ağında da kullanılacak) ile karışmayı engeller.

Yukarıda yapılan tanımlar Petri ağlarındaki durum geçiş mekanizmalarını açık olarak ifade etmez. Petri ağlarının dinamik DES'lerin modellenmesinde kullanılması istendiği için bu nokta önemlidir. Durum geçiş mekanizmasına Petri ağ grafının yapısından giderek ulaşılır. Temel olarak, $t \in T$ geçişi için "oluştu" veya "mümkün oldu" işlemini tanımlamak için, geçişe giriş olan her yerde bir jeton olması gereklidir.

Tanım 2.3 "İzinli Geçiş":

Bir Petri ağında $t_j \in T$ olmak üzere eğer

$$x(p_i) \geq w(p_i, t_j) \qquad \text{tüm } p_i \in I(t_j) \text{'ler için} \qquad \textbf{(2.9)}$$

koşulu sağlanıyorsa t_j geçişi "izinlidir" denir. Kelimelerle açıklamak gerekirse, t_j geçişine giriş olan tüm p_i yerleri için, p_i'de bulunan jeton sayısı p_i'yi t_j'ye bağlayan arkların ağırlığına eşit veya büyük olduğunda, t_j geçişi "izinlidir" denir.

Şekil 2.3'teki x_1 durumunda, $x(p_1) = 1 < w(p_1, t_1)$ olduğundan t_1 geçişi izinli değildir. Fakat x_2 durumunda $x(p_1) = 2 = w(p_1, t_1)$ olduğundan t_1 geçişi izinlidir.

Daha önce bahsedildiği gibi, yerler bir geçişin izinli olabilmesi için gerekli koşullarla ilişkili olduğundan bir geçiş bütün koşullar sağlandığında olabilecektir. Jetonlar sağlanması gereken koşulları belirlemek için kullanılan bir mekanizmadır. Bir Petri

ağının verilen bir durumunda izinli geçişlerinin kümesi, otomatlardaki bir durumun aktif olay kümesine eşdeğerdir.

2.2.3 Petri Ağ Dinamikleri

Otomatlarda durum geçiş mekanizması, durumları(düğümler) birbirine bağlayan arklar ile ve eşdeğer olarak f geçiş fonksiyonları ile doğrudan durum geçiş diyagramında ortaya çıkar. Petri ağlarında ise durum geçişleri graf üzerinde görülmezler. Ancak jeton hareketi ile bir durumdan diğerine geçiş gösterilebilir. Bir geçiş izinli olduğu zaman "olay oluştu" ya da "ateşlenebilir" (Petri ağ literatüründe) deyimi kullanılır. Bir Petri ağının durum geçiş fonksiyonu, izinli geçişlerin ateşlenmesine bağlı olarak Petri ağının durumunda oluşan değişme üzerinden tanımlanır.

Tanım 2.4 "Petri Ağ Dinamikleri":

(P,T,A,w,x) Petri ağının durum geçiş fonksiyonu $f : N^n \times T \to N^n$, $t_j \in T$ geçişi için ancak ve ancak

$$x(p_i) \geq w(p_i,t_j) \qquad \text{tüm } p_i \in I(t_j) \text{'ler için} \qquad \textbf{(2.10)}$$

sağlanıyorsa tanımlanır.

Eğer $f(x,t_j)$ tanımlı ise, $x' = f(x,t_j)$ yeni durumu,

$$x'(p_i) = x(p_i) - w(p_i,t_j) + w(t_j,p_i), \qquad i = 1,...,n. \qquad \textbf{(2.11)}$$

ile tanımlanır. (2.11) denklemi yerlerdeki yeni jeton sayılarını belirler, zaten durum değiştirme jeton dağılımının yerlerde değişmesi anlamına gelir.

(2.10) koşulu sadece izinli geçişler için durum geçiş fonksiyonunun tanımlı olmasını sağlar, bir izinli geçiş otomattaki "mümkün olay (feasible event)"a eşdeğerdir. İzinli geçişler hangi yerlerden izin alıyorsa o yerlerdeki jetonların sayısı değişir, dolayısıyla durum değişikliği olur. Burada durumlar yerlerde jetonlar yardımıyla bir nevi kodlanmıştır ve yerlerdeki jeton sayısı değiştikçe durum değişikliği olur. Otomatlarda durum geçiş fonksiyonu keyfi olmasına rağmen Petri ağında Petri ağının yapısına dayanır. Böylece (2.11) ilişkisi ile verilen bir sonraki durum, bir geçişin giriş ve çıkış yerlerine ve de geçişi bu yerlere bağlayan arkların ağırlıklarına bağlıdır.

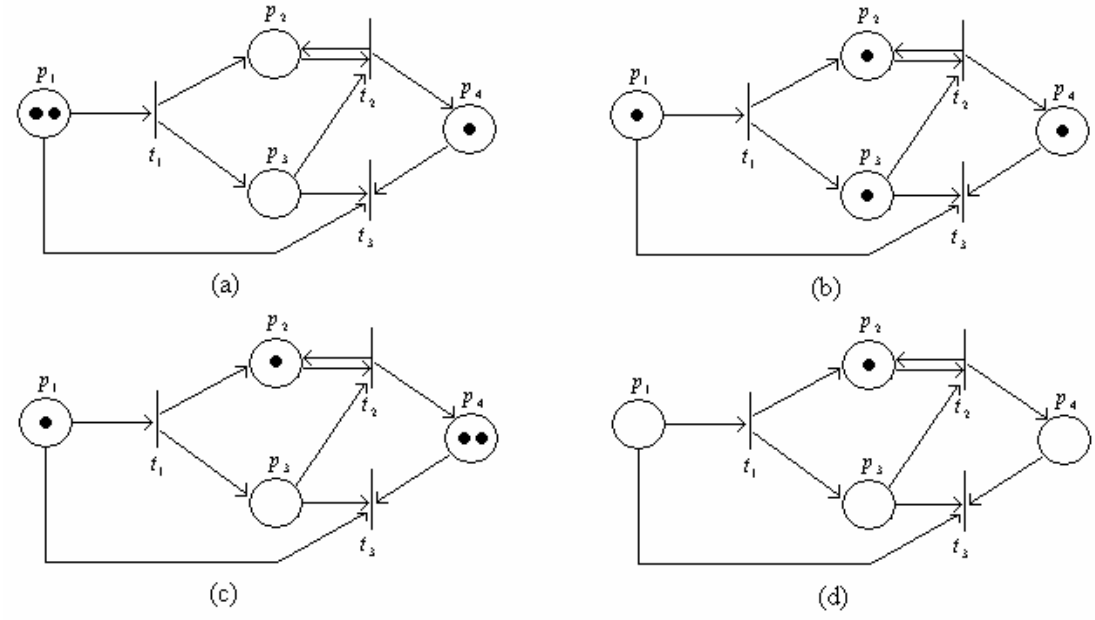
(2.11)'e göre p_i, t_j 'nin giriş yeri ise p_i'yi t_j 'ye bağlayan arkın ağırlığı kadar jeton kaybedecek, çıkış yeri ise t_j 'yi p_i'ye bağlayan arkın ağırlığı kadar jeton kazanacaktır. Bazı yerler (p_i) hem giriş hem de çıkış yeri özelliğinde olabilir, bu durumda $w(p_i, t_j)$ jeton p_i'den alınacak ve hemen $w(t_j, p_i)$ jeton p_i'ye geri verilecektir.

Şuna dikkat edilmelidir ki, Petri ağında ateşlenen bir geçişin, öncesi ve sonrasındaki jeton sayısının sabit kalması gerekli değildir. Daha açık ifade etmek gerekirse,

$$\sum_{p_i \in P} w(t_j, p_i) > \sum_{p_i \in P} w(p_i, t_j) \quad \text{veya} \quad \sum_{p_i \in P} w(t_j, p_i) < \sum_{p_i \in P} w(p_i, t_j) \qquad \textbf{(2.12)}$$

ifadeleri geçerlidir. Dolayısıyla $x' = f(x, t_j)$ x'den fazla veya az jeton bulundurabilir. Böylece bir takım geçiş ateşlemelerinden sonra jeton sayılarının yerlerdeki dağılımı olarak tanımlanan $x = [0, 0, ..., 0]$ veya bir yerdeki jeton sayısının çok arttığı değerlere ulaşılabilir. Bu ise bu şekilde tanımlanan x'lerin sayılarının sonsuz olabileceğini gösterir.

Örnek 2.4: (Geçişlerin Ateşlenmesi)



Şekil 2.4 : Petri Ağının Geçiş Ateşlemeleri

Şekil 2.4'te bir petri ağındaki geçiş ateşlemeleri sonucu değişen durumlar görülmektedir. Şekil 2.4 (a)'dan görüldüğü gibi başlangıç durumu $x_0 = [2,0,0,1]$'dir. Bu şekilde izinli tek geçiş t_1'dir. $x_0(p_1) = 2 \geq 1 = w(p_1, t_1)$ koşulunu sağladığı görülür. t_1 ateşlendiğinde p_1'den bir jeton alınır, p_2 ve p_3 yerlerine bir jeton konur. (2.11) denklemi uygulandığında da gelinen yeni durumun $x_1 = [1,1,1,1]$ olduğu görülür. Şekil 2.4 (b) yeni durumu göstermektedir. Bu durumda 3 geçiş te (t_1, t_2, t_3) izinlidir. t_2'nin ateşlendiği düşünülürse giriş yerleri p_2 ve p_3'ten birer jeton alınmalıdır. Çıkış yerleri p_2 ve p_4'tür, bu nedenle p_2'den alınan jeton geri konulur, $p_2 \in I(t_2) \cap O(t_2)$. p_4'e bir jeton eklenir. Yeni durum Şekil 2.4 (c) ile gösterilen $x_2 = [1,1,0,2]$'dir. Bu durumda t_2 ve t_3 geçişleri artık izinli değildir, t_1 geçişi hala izinlidir.

Eğer Şekil 2.4 (b)'deki x_1 durumuna dönülüp t_2 yerine t_3 ateşlenirse; p_1, p_3 ve p_4 giriş yerlerinden birer jeton alınır. Çıkış yerinin olmadığına dikkat edilmelidir. Yeni gelinen durum Şekil 2.4 (d) gösterilen $x_2' = [0,1,0,0]$ durumudur. Bu durumda hiçbir geçişin izinli olmadığı görülür. Herhangi bir durum değişimi mümkün değildir ve $[0,1,0,0]$ durumuna Petri ağının "açmaz (deadlock)" durumu denir.

Şekillerden görüldüğü gibi jeton sayıları korunmamıştır. x_0 durumunda 3 jeton varken x_1 ve x_2'de 4, x_2''te 1 jeton vardır.

Bu örnek, geçişlerin sırasının önceden belirlenmediği bir Petri ağında durum geçişlerini göstermiştir. x_1 durumunda tüm geçişler (3 geçişten herhangi biri) ateşlenebilir. Bu durum, DES'in otomat modelinde x_1 durumunun aktif olay kümesinde 3 olayın da olması gibi düşünülebilir. Aslında bir Petri ağı için bu her durum için söylenebilir, yani Petri ağında tüm olaylar her durumda aktif olay kümesine konabilir ama izinli olmayanlar kendi üzerinde öz çevrimlerle bağlanmış gibi düşünülebilir, ya da ileride tanımlanacak "ateşleme vektörü" giriş olarak alınabilir.

Diğer bir önemli gözlem ise Petri ağlarının dinamik davranışı ile ilgilidir. Verilen bir ilk koşul için bir Petri grafının N^n durumlarından hepsine erişmenin gerekli olmadığıdır. Şekil 2.3'teki $x_0 = [2,1]$ durumu ele alınırsa, $x_1 = [0,3]$ olur ve bu başlangıç durumu için erişilebilen tek durum budur. Bu durum sonucunda "erişilebilir durumlar" tanımının yapılması uygun olur. (P, T, A, w, x) Petri ağının erişilebilir durumlar kümesi $R[(P, T, A, w, x)]$ olarak tanımlanır. Buna bağlı olarak, önce f durum geçiş fonksiyonunun tanım kümesinin $N^n \times T \to N^n \times T^*$ şeklinde genişletilmesi gerekir. Bu otomatlarda da yapılmıştır.

$$f(x,\varepsilon)=x \tag{2.13}$$

$$f(x,st)=f\left(f(x,s),t\right) \qquad s\in T^{*} \text{ ve } t\in T \text{ için} \tag{2.14}$$

Burada ε sembolünü geçiş ateşlemesinin var olmadığı olarak yorumlamak gerekir.

Tanım 2.5 "Erişilebilir Durumlar":

(P,T,A,w,x) Petri ağı için "Erişilebilir Durumlar"

$$R\left[(P,T,A,w,x)\right]=\left\{y\in N^{n}:\exists s\in T^{*}\left(f(x,s)=y\right)\right\} \tag{2.15}$$

olarak tanımlanır.

Yukarıda yapılan erişilebilir durumlar kümesi R ve genişletilmiş biçimdeki durum geçiş fonksiyonu f tanımlarında, eş zamanlı ateşlemeye izin verilmediği sadece birinin ateşlendiği varsayılmıştır. Önceki örnekte, Şekil 2.4 (b) durumuna bakılırsa t_1, t_2 ve t_3'ün hepsi izinli geçiştir. Eğer t_1 ve t_2'nin aynı anda ateşlendiği düşünülseydi farklı durumlar elde edilebilirdi. Daha sonra etiketli geçişlerden, mümkün tüm erişilebilir durumlardan ve bunlardan yola çıkarak da bir Petri ağı tarafından üretilen ve işaretlenen dillerden söz edilebilmesi için eş zamanlı ateşleme dışarıda bırakılacak, geçişlerin birer birer ateşlendiği varsayılacaktır.

2.2.3.1 Durum Denklemleri

(2.11) denklemine dönülürse, bu kural gereği $x(p_i)\in N$'den $x'(p_i)\in N$'ye geçildiği görülür. Bunlar N^n'de tanımlı vektörler olarak aşağıdaki gibi gösterilebilir.

$$x'=\left[x'(p_1),x'(p_2),...,x'(p_n)\right] \tag{2.16}$$

$$x=\left[x(p_1),x(p_2),...,x(p_n)\right] \tag{2.17}$$

Özel bir geçiş t_j'nin ateşlenmesinin kuralı da m boyutlu satır vektörü olan u "ateşleme vektörü" ile aşağıdaki gibi tanımlanabilir.

$$u=[0,...,0,1,0,...,0] \tag{2.18}$$

$j.$ geçişin ateşlendiğini belirtmek için sadece $j.$ eleman "1" olur. $j \in \{1,...,m\}$ 'dir. Daha önce belirtilen birer birer ateşleme varsayımı sonucunda aynı anda sadece bir vektör elemanı "1" olacaktır. Ek olarak, bir Petri ağının ağ matrisi A, $m \times n$ boyutunda olup $(j,i).$ elemanı

$$a_{ji} = w(t_j, p_i) - w(p_i, t_j) \tag{2.19}$$

t_j'ye gelen ve çıkan ark sayısının farkı ya da arkların ağırlıklarının farkı ile hesaplanır. Bu durumda yeni durumu veren ifade

$$x' = x + uA \tag{2.20}$$

şeklinde yazılabilir ve burada u giriş gibi düşünülebilir. Bu denklem

$$f(x, t_j) = x + uA \tag{2.21}$$

şeklinde de yazılabilir. Burada t_j argümanı u'nun "1" olan $j.$ elemanını belirtir.

Böylece geçiş ateşleme prosesi ve Petri ağının durum değişimleri için grafik gösterilim dışında cebirsel gösterilim elde edilmiş olur.

Örnek 2.5: (Durum Denklemi)

Başlangıç durumu $x_0 = [2,0,0,1]$ olan Şekil 2.4 (a) grafı tekrar ele alınsın. Öncelikle graf incelenerek A matrisi aşağıdaki şekilde yazılabilir.

$$A = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & -1 & -1 \end{bmatrix}$$

Örneğin (1,2). eleman $w(t_1, p_2) - w(p_2, t_1) = 1 - 0 = 1$ şeklinde hesaplanmıştır. Bu hesaplamalar tüm geçiş ve yerler için yapılarak A matrisi oluşturulmuştur. (2.20) denklemi kullanılarak başlangıç durumunda t_1 geçişi ateşlendiğinde,

$$x_1 = [2 \quad 0 \quad 0 \quad 1] + [1 \quad 0 \quad 0] \begin{bmatrix} -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & -1 & -1 \end{bmatrix}$$

$$= [2 \quad 0 \quad 0 \quad 1] + [-1 \quad 1 \quad 1 \quad 0] = [1 \quad 1 \quad 1 \quad 1]$$

Örnek 2.4'teki x_1 durumu elde edilmiş olur. Benzer şekilde t_2 geçişinin ateşlenmesiyle x_2 durumu,

$$x_2 = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} + \begin{bmatrix}0 & 1 & 0\end{bmatrix}\begin{bmatrix}-1 & 1 & 1 & 0\\ 0 & 0 & -1 & 1\\ -1 & 0 & -1 & -1\end{bmatrix}$$
$$= \begin{bmatrix}1 & 1 & 0 & 2\end{bmatrix}$$

şeklinde elde edilir. Bu şekilde jetonların yerlerdeki dağılımı ve erişilebilir durumlar ortaya çıkmış olur.

Dinamik bir sistemin modeli olarak, Petri ağı otomatlardakine benzer olarak bir yörünge ortaya çıkarır. Yörünge bir durumlar dizisidir. $\{x_0, x_1, x_2, ...\}$ yörüngesi $\{t^1, t^2, ...\}$ gibi bir giriş yörüngesine karşı düşer. Burada t^k $k.$ geçişin ateşlenmesidir. Bu nedenle

$$x_{k+1} = f(x_k, t_k) = x_k + u_k A \qquad \textbf{(2.22)}$$

yazılabilir. u_k $k.$geçişin ateşlendiği bilgisini içerir. Eğer artık hiçbir ateşlemenin mümkün olmadığı duruma gelinmişse $x_{k+1} = x_k$ olur ve bu açmaz durumdur.

2.2.4 Petri Ağ Dilleri

Geçişler = olaylar karşı düşürmesi yapılarak bir dilden söz etmek mümkündür. Ancak bu karşı düşürme kesin bir yargı değildir yani eğer bir dili göstermek için Petri ağı bir modelleme formalizmi olarak seçilmişse ya da Petri ağına bu açıdan bakılıyorsa her geçişe bir olay karşı düşürmeye gerek vardır. Bu yolla bir Petri ağı tarafından üretilmiş ve işaretlenmiş diller tanımlanabilir.

E olaylar kümesi ile verilmiş ve dili bir Petri ağı ile modellenmiş bir DES gözönüne alınsın. $T = E$ olacaktır ve her geçişe bir olay karşı düşürülecektir. Ancak bu gereksiz yere yapılmış bir kısıt olabilir, bu nedenle iki arkın aynı olayla ilişkilendirilmiş olduğu duruma izin verilir. Buna "etiketlenmiş Petri ağı" denir.

Tanım 2.6 "Etiketlenmiş Petri Ağı":

Etiketlenmiş Petri ağı N 8 elemanlı olup

$$N = (P, T, A, w, E, l, x_0, X_m) \qquad \textbf{(2.23)}$$

ile gösterilir. Burada,

(P,T,A,w) Petri ağ grafını,

E, geçişlere etiketlenmiş olaylar kümesini,

$l:T\to E$, geçişlerin etiketlenme fonksiyonunu,

$x_0\in N^n$, ağın başlangıç durumunu,

$X_m\subseteq N^n$, ağın işaretli durumlarını gösterir.

Petri ağ grafında bir geçişin etiketi geçişin üstünde gösterilir. İşaretli durumlar kavramı tamamen otomattaki ile aynıdır. İşaretli durumlar, etiketli Petri ağının işaretlediği dili belirlemekte kullanılacaktır.

Tanım 2.7 "Üretilen ve İşaretlenen Dil":

$N=(P,T,A,w,E,l,x_0,X_m)$ etiketlenmiş Petri ağı tarafından üretilen dil,

$$L(N)=\{l(s)\in E^*:s\in T^* \text{ ve } f(x_0,s) \text{ tanımlı}\} \qquad \textbf{(2.24)}$$

ile ve işaretli dil,

$$L_m(N)=\{l(s)\in L(N):s\in T^* \text{ ve } f(x_0,s)\in X_m\} \qquad \textbf{(2.25)}$$

ile gösterilir. Burada $l:T^*\to E^*$ tanımlı olarak genişletilmiştir. Görüldüğü gibi bu tanım, otomatta yapılan tanım ile tamamen tutarlıdır. $L(N)$, N'deki geçiş ateşlemelerinin mümkün bütün sonlu dizilerinden elde edilen geçiş etiketlerinin tüm kelimelerini gösterir. Geçişlere etiketlenmiş tüm kelimeler de denilebilir. $L(N)$ ilk koşul x_0'a bağlıdır. $L_m(N)$ ise $L(N)$'deki kelimelerin bir alt kümesidir ve özel olarak x_0'dan X_m işaretli durumlar kümesine götüren geçişler dizisinin karşı düştüğü kelimeler kümesidir.

Bir etiketlenmiş Petri ağının ifade edebildiği dilin özelliği,

$$PNL=\{K\subseteq E^*:\exists N=(P,T,A,w,E,l,x_0,X_m)[L_m(N)=K]\} \qquad \textbf{(2.26)}$$

ile gösterilir. Bu genel bir tanımdır. *PNL*'nin özellikleri ağırlıklı olarak yapılan varsayımlara bağlıdır (l'nin tek yönlü olup olmadığı, X_m'in sonlu olup olmadığı

gibi.). Ancak l'nin tek yönlü olma ve X_m'in sonlu olma koşulu yoktur. Her zaman böyle bir dil tanımlanabilir.

2.2.5 Kuyruk Sistemleri İçin Petri Ağ Modelleri

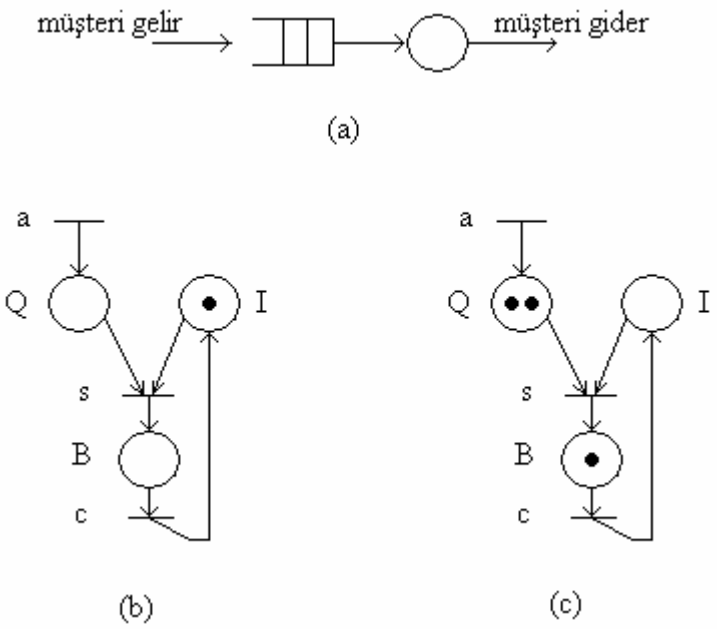
Bir kuyruk sisteminin dinamik davranışı Petri ağ yapısı kullanılarak gösterilecektir. 3 olaylı (geçişler) bir sistem ele alınsın.

a: müşteri gelir

s: servis başlar

c: servis biter ve müşteri gider.

Burada geçiş kümesi $T = \{a, s, c\}$'dir. Bu örnekte etiketlenmiş Petri ağlarını düşünmeye gerek yoktur, başka bir deyişle $E = T$ olduğu varsayılabilir. a geçişi spontane olup koşul gerektirmez (giriş yeri yoktur). Diğer taraftan s geçişi iki koşula bağlıdır; kuyruktaki müşterilerin varlığı ve servisçinin boş olması. Bu koşullar Q (kuyruk) ve I (boş servisçi) giriş yerleri ile gösterilir. Son olarak da c geçişi servisçinin meşgul olmasını gerektirir ve bunun için B (meşgul servisçi) giriş yeri tanımlanır. Böylece $P = \{Q, I, B\}$ yer kümesi elde edilir.



Şekil 2.5 : Kuyruk Sisteminde Petri Ağ Modelleri

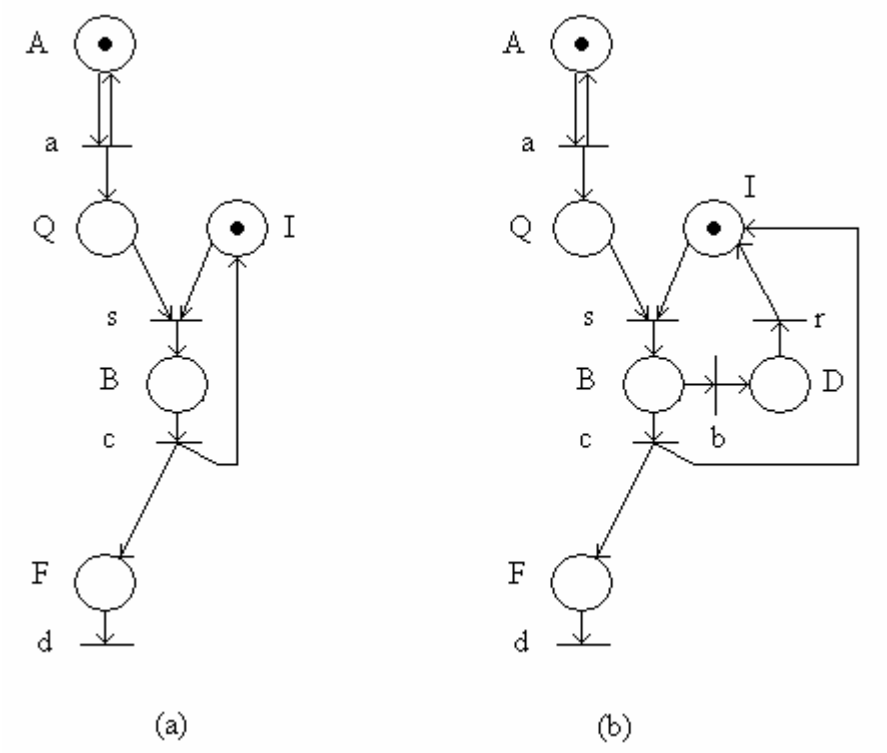
Basit kuyruk sistemini modelleyen Petri ağ grafı Şekil 2.5 (a) ve Şekil 2.5 (b) ile gösterilmektedir. Q'da hiç jeton yoktur, kuyruğun boş olduğunu gösterir ve I'da bir

jeton vardır, servisçinin boş olduğunu gösterir. Başlangıç durumu $x_0 = [0,1,0]$'dır. a geçişi her zaman izinlidir. Şekil 2.5 (c) $\{a,s,a,a,c,s,a\}$ geçişleri ateşlendikten sonra gelinen $[2,0,1]$ durumunu gösterir. Bu durum, kuyrukta 2 müşterinin beklediğini ve 3.'nün serviste olduğunu gösterir. İlk gelen müşteri c geçişinden sonra gitmiştir.

Bir c geçişi her zaman s geçişini izinli yapar çünkü Q yerinde bir jeton bulunur. Aynı kuyruk sisteminin biraz daha ayrıntılı modeli

d: müşteri gider.

geçişi eklenerek sağlanabilir. Bu durum F (bitiren müşteri) koşulunu gerektirir. C geçişi sadece "servis biter" anlamına gelir. Ek olarak a geçişine giriş yeri olarak A tanımlanır, a geçişini izinli kılmak için burada her zaman bir jeton bulundurulur. Böylece bu alternatif modelde $T = \{a,s,c,d\}$ ve $P = \{A,Q,I,B,F\}$ olur. Sonuç model $[1,0,1,0,0]$ durumu ile Şekil 2.6 (a)'da gösterilmektedir.



Şekil 2.6 : Kuyruk Sisteminde Alternatif Petri Ağ Modelleri

Şekil 2.6 (a)'daki Petri ağ modeli, servisçinin bozulması durumu gözönüne alınarak daha da değiştirilebilir. Bu durumda iki yeni geçiş tanımlanır.

b: servisçi bozulur

r: servisçi tamir edilir.

r geçişine giriş yeri olarak servisçinin bozuk olduğu durumu gösteren D (servisçi bozuk) tanımlanır. Böylece $T=\{a,s,c,d,b,r\}$ ve $P=\{A,Q,I,B,F,D\}$ olur. Bu Petri ağ modeli Şekil 2.6 (b)'de $[1,0,1,0,0,0]$ durumu için gösterilmiştir.

2.3 Petri Ağları ve Otomatların Karşılaştırılması

Otomat ve Petri ağı her ikisi de bir DES'in davranışını modellemek için kullanılırlar. Her iki formalizm de DES'in durum geçiş yapısını açık olarak gösterir. Otomatta bu iş için, mümkün bütün durumlar numaralandırılır ve bu durumlar mümkün olan tüm geçişlerle birbirine bağlanır, böylece bu gösterilim otomatın geçiş fonksiyonlarıyla sonuçlanır. Bu sadece özel olarak şık bir gösterilim değil aynı zamanda "çarpma" ve "paralel birleştirme" gibi birleştirme işlemleriyle donatılabildiği için karmaşık sistemleri oluşturan bileşenlerin modellenmesi ve sistematik bir birleştirme işlemi ile karmaşık sistemi modellemeyi kolaylaştırır. Petri ağları ise geçiş fonksiyonlarının yapılarını daha fazla öne çıkaran bir yapıya sahiptirler. Durumlar numaralandırılmazlar. Aslında durum bilgisi graf içine dağılmıştır ve geçişler sonunda ortaya çıkar.

Şu beklenen bir sorudur: "Verilen bir DES'in modellenmesinde hangi model daha iyidir? Otomat mı Petri ağı mı?" Böyle bir sorunun aşikar bir cevabı yoktur. Modelleme genellikle kişisel tercihlere ve sıklıkla da özel uygulamalara bağlıdır. Buna rağmen, eğer yukarıdaki soru, karşılaştırma için belirli kriterler bağlamında daha kesin formüle edilirse, bazı sonuçlar vermek mümkün olabilir.

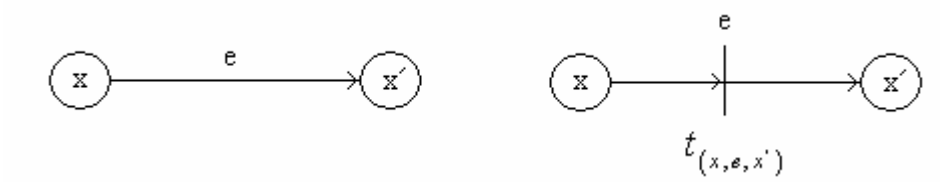
2.3.1 Dilin İfade Edilebilirliği ve Otomattan Petri Ağına Geçiş

Otomatlar ve Petri ağlarının karşılaştırılmasında ilk kriter olarak her iki formalizmin gösterebildiği dillerin karşılaştırması yapılacaktır. Bu önemlidir çünkü sonlu hafıza pratik bir sınırlamadır. Kesin olarak PNL'nin R'den daha geniş bir sınıf olduğu iddia edilir, bunun anlamı sonlu yerler ve geçişlerle ifade edilen E^*'ın içindeki dillerin sayısı sonlu durumlu otomatlardan daha fazladır. Bunu ispat edebilmek için önce, bir sonlu durumlu otomatın her zaman bir Petri ağı karşılığının nasıl elde edildiği görülecektir. Bu yapıldığında, R ile gösterilen bütün regüler dillerin bir Petri ağı tarafından işaretlendiği gösterilmiş olur. Daha sonra ispatı tamamlamak için bir regüler olmayan dili işaretleyen Petri ağ bulunması yeterli olur.

$G=(X,E,f_G,\Gamma,x_0,X_m)$ sonlu durumlu otomatı verilmiş olsun. Bu durumda X ve

E sonlu bir küme olacaktır. $L(N)=L(G)$ ve $L_m(N)=L_m(G)$ olan $N=(P,T,A,w,E,l,x_0,X_m)$ Petri ağı aşağıdaki adımlar izlenerek oluşturulacaktır.

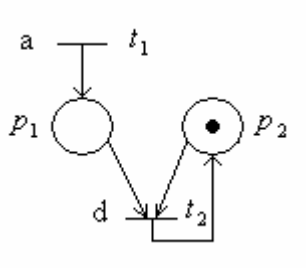
1. Her durum ($\in X$) bir yer ($\in P$) ile tanımlansın yani $P=X$ olsun.
   a. N'in x_0 durumu bir nötr vektördür, $[0,...,0,1,0,...,0]$. "0"dan farklı olan eleman $x_0 \in X$ durumuna karşı düşen yer ($\in P$) içindir.
   b. X_m'ler de benzer şekilde oluşturulurlar. Önce $X \to P$'ye , $X_m \to P_m$'e ve bunlara karşı düşenler N'in X_m'leri olarak $[x(p_i)]$ olarak oluşturulur.
2. G'deki her üçlü (x,e,x') $x'=f_G(x,e)$ ($e \in \Gamma(x)$ aktif olay kümesi) bir geçiş ile gösterilir. Bu $t_{(x,e,x')} \in N$ olacaktır. Diğer bir deyişle T'nin kardinalitesi, G'nin durum geçiş diyagramındaki arkların kümesinin kardinalitesi ile aynı olacaktır.
   a. T'deki $t_{(x,e,x')}$ geçişi $e \in E$ olayı ile etiketlensin;
   b. G'deki her üçlü (x,e,x') için A'da iki ark tanımlansın: $arc(x,t_{(x,e,x')})$ ve $arc(t_{(x,e,x')},x')$. Bütün bu arklar eşit ve "1" ağırlığında olsun. Bu işlem aşağıdaki şekilde oluşturulur.



Şekil 2.7 : Otomattan Petri Ağına Geçiş

Burada verilen yöntem gerçekte gereksiz uzun bir yöntemdir, sadece bu ispatın yapılması için gereklidir. Gerçekte çok daha kolay yoldan bir otomattan bir Petri ağına geçilebilir.

PNL'in R'den geniş olduğunu göstermek için tekrar Şekil 2.5'teki kuyruk sistemi yapısına geçilir, B yeri ve c geçişi çıkarılırsa Şekil 2.8'deki Petri ağı elde edilir.



Şekil 2.8 : Şekil 2.5 (a) Kuyruk Sisteminin Düzenlenmesi

Burada "a: kuyruğa bir kişi geldi" ve "d: servis başladı" anlamındadır.

Bu durumda bu Petri ağının işaretlediği dil regüler olmaz çünkü x'in sayısı ∞'a ulaşır. Çünkü $x(p_1) \in N$ olacaktır ve keyfi olarak büyüyebilecektir. Buna rağmen Petri ağ grafı sonlu yer ve geçişe sahip olur ve sorun yaratmaz.

Bütün Petri ağlarına karşı düşen bir otomat bulunması mümkün olmazken erişilebilir durumlarının sayısı sonlu $R(N)$ olan bir Petri ağına karşı düşen bir otomat bulunabilir. $x \in R(N)$ için her durumda mümkün geçişlere karşı düşen arklar yardımıyla bir otomat kolaylıkla elde edilebilir.

Bir Petri ağının erişilebilir durumlarının sayısı sonsuz olabildiği için, bir otomatın işaretlediği ya da üretebildiği dilden daha geniş bir dil ailesini işaretler.

Bu çalışmanın uygulama kısmında, girilen bir işaretli dilin $G = \{X, E, f, \Gamma, x_0, X_m\}$ otomatının durum geçiş diyagramından $N = (P, T, A, w, E, l, x_0, X_m)$ Petri ağına geçilmiş, durum denklemleri kullanılarak oluşan olaylara karşı düşen yeni durumlar hesaplanmıştır.

2.3.2 Modüler Model İnşaa Etme

Otomatlarda etkileşimli olarak çalışan X_1 durumlu bir alt otomat, X_2 durumlu bir başka otomat ile bağlandığı zaman durum uzayı $X_1 \times X_2$'ye artar. Bu yeni elde edilen otomatın d_1 sayısının aşırı artması anlamına gelir yani karmaşıklaşması demektir. Diğer taraftan Petri ağ modelinde ise bu bağlantıyı yapmak daha kolaydır. Bu işlem birkaç yer veya geçiş ekleyerek veya birkaç yeri değiştirerek yapılabilir. Ayrıca bir Petri ağ grafına bakarak özel bileşenleri uygun olarak ayırmak, bunların arasındaki etkileşimlerin seviyesini fark etmek ve sistemi farklı modüllere ayırmak mümkündür. Kuyruk sistemi örneği ile bu iddialar gösterilmiştir. Ayrıca sistem bileşenleri modelinin bir lineer kombinasyonu olarak tüm sistem oluşacaktır yani büyüme eksponansiyel olmayacaktır. Oysa "çarpma" ve "paralel birleştirme" ile

bağlamada çarpımsal bir büyüme söz konusudur. Gerçekte "çarpma" ve "paralel birleştirme" ile bağlamayı Petri ağı için de tanımlamak mümkündür. Ancak bu şekilde yapılacak olan bir birleştirme işleminin sonucunda oluşan sistemin genel olarak karmaşıklığının lineer olarak arttığından söz etmek mümkün olmaz.

2.3.3 Karar Verilebilirlik

Otomatlar ve Petri ağları için bir başka karşılaştırma kriteri "karar verilebilirlik"tir. Karar verilebilirlik, "Bu olay dizisi bu sonlu durumlu otomat tarafından tanınabilir mi?" sorusuna "Evet" veya "Hayır" diyebilmenin sistematik bir yolunun olup olmaması olarak ele alınır. Sonlu durumlu bir otomatın çekici taraflarından biri de, bu sorunun otomat için kesin olması ve "karar verilebilir" bir yapı olmasıdır. Maalesef bu soru her zaman Petri ağı için doğru değildir, karar verilebilirlik ve model zenginliği arasındaki doğal alışverişin yansımasıdır.

Tüm bu karşılaştırmalardan çıkan sonuç, belki de Petri ağ ve otomatların birbirleriyle rekabet eden değil birbirlerini tamamlayan modelleme yaklaşımı olmasıdır. Özellikle bazı uygulamalar için uygunluğuna göre biri tavsiye edilebilir.

2.4 Petri Ağlarının Analizi

Petri ağ modellerinin problemleri Bölüm 2.4.1'de sınıflandırılacaktır. Bu problemler özellikte otomatta "güvenlik" ve "kilitlenme (blocking)" olarak ele alınan konularla ilgilidir. Bununla birlikte, Petri ağ modellerindeki yapısal bilgiler, bu problemlerin daha spesifik versiyonlarını sorgulamak için kullanılır. Bunlar; sınırlılık, sakınım, kapsanabilirlik, engellenemez olmadır.

2.4.1 Problemlerin Sınıflandırılması

Bu bölümde ele alınan konular, Petri ağların lojik davranışı ile ilgilidir. Bu konular öncelikle istenen özellikler ile bağlantılıdır. Bu özelliklerin pek çoğunun tanımlanma nedeni, Petri ağlarının çevredeki kaynakların paylaşımında kullanılması, kaynakların verimli ve adil kullanılmasının istenmesidir.

2.4.1.1 Sınırlılık

Bir çok durumda, jetonlar, kaynak paylaşımlı sistemde müşterilere karşı düşer. Örneğin Şekil 2.5'te Q yerindeki jetonlar kuyruğa giren müşterileri göstermektedir. Elbette burada kuyruktaki müşteri sayısının ∞'a artması istenmeyen bir durumdur.

Klasik sistemlerde bir durum değişkeninin ∞'a artması "kararsızlık"a karşı düşer. Benzer şekilde burada da bir durum değişkenindeki sınırsız artış "kararsızlık formu"na gitmeye neden olur.

Sınırlılık, bir yerde bulunan jeton sayısının verilen pozitif bir sayıyı geçmemesi anlamına gelir.

Tanım 2.8 "Sınırlılık":

x_0 ilk koşulu ile verilmiş bir Petri ağında, bir $p_i \in P$ yerinin "k-sınırlı" veya "k-güvenli" olma tanımı $x \in R(N)$ tüm erişilebilir durumlar için bir p_i yerindeki jeton sayısının k ile sınırlandırılmış olmasıdır yani $x(p_i) \leq k$.

Eğer yer 1-sınırlı ise "güvenli"dir denir. Önceden belirlenmiş olmasına gerek olmayan bir sayı k olmak üzere, bir yer k-sınırlı ise "sınırlı"dır denir. Eğer Petri ağındaki tüm yerler sınırlı ise ağ "sınırlıdır" denir. Şekil 2.5'te verilen modelde Q'daki müşteri sayısı keyfi artabileceği için bu ağ "sınırlı" değildir.

Bir Petri ağı ile modellenmiş bir DES verildiğinde, sınırlılığa bakılır ve eğer "sınırlı" ise sınır bulunur. Eğer sınırlılık sağlanmıyorsa, modelin sınırlılığı sağlayacak şekilde değiştirilmesi düşünülür. Eğer Petri ağı sınırlı ise, istenirse daha önce bahsedildiği gibi bir otomat modeline geçilebilir ve oradaki analiz tekniklerinden yararlanılabilir.

2.4.1.2 Güvenlik ve Kilitlenme

Bu konu ya durumlar ya da diller üzerinden ele alınabilir. Yani ya duruma erişilebilirlik ya da alt kelimeler ile ilgilenilir. Eğer bir Petri ağı "sınırlı" ise güvenlik ve kilitlenme özellikleri algoritmik olarak belirlenebilir. Eğer bir Petri ağı 0 ve 1 yer işaretlemelerinden oluşan durumlar içeriyorsa bu ağ "güvenli"dir denir.

2.4.1.3 Durumun Kapsanabilirliği

"Durum kapsanabilirliği", durumların erişilebilirlik kavramının bir genelleştirilmesidir. Aynı zamanda, özel bir geçişin ateşlenebilme kavramı ile de ilişkilidir. Bir geçişin izinli olması için bazı yerlerde belirli bir sayıda jeton olması gerekir. Örneğin, $y = [y(p_1), y(p_2), ..., y(p_n)]$ durumu verilsin ve t_j geçişinin izinli olması için gereken özelliği yani jeton sayısı açısından sağlasın. x_0 durumunda bulunulsun ve buradan bakıldığında t_j'nin izinli olduğu görülmek istensin. Bu

durumda x_0'dan kalkılıp $x(p_i) \geq y(p_i)$ $i = 1,2,...,n$ koşulunu sağlayan bir x'e gidilip gidilemeyeceğinin bilinmesi gerekir. Eğer bu oluyorsa (yani $x(p_i) \geq y(p_i)$ x_0 başlangıç durumlu bir Petri ağı için mümkün ise) "x, y durumunu kapsar" denir.

Tanım 2.9 "Durumun Kapsanabilirliği":

x_0 ile verilmiş bir Petri ağ için, eğer $x(p_i) \geq y(p_i)$ özelliğinde bir $x \in R(N)$ durumu varsa "y bu ağ tarafından kapsanabilirdir" denir.

2.4.1.4 Sakınım

Sakınım, Petri ağının bir özelliğidir. Bir yörünge boyunca ulaşılan tüm durumlar için "sabit sayıda jeton" sağlama olarak tanımlanır. Ancak bu çok sınırlayıcı bir özellik olacaktır.

Formal bir tanım yapmak için $\gamma = [\gamma_1, \gamma_2, ..., \gamma_n]$ $\gamma_i \geq 0$ tanımlansın. γ_i'ler p_i yerlerinin ağırlığı olarak düşünülebilir. γ_i'ler tamsayı olarak alınacaktır, bu gereklilik değildir, basitlik için yapılmaktadır.

Tanım 2.10 "Sakınım":

x_0 ilk koşullu Petri ağı, bir $\gamma = [\gamma_1, \gamma_2, ..., \gamma_n]$ ve tüm $x \in R(N)$ için, $\sum_{i=1}^{n} \gamma_i x(p_i) =$ sabit ise "γ'ya göre sakınım özelliğindedir" denir.

Verilen bir Petri ağ modeli için, sıklıkla kaynakların yok olması veya kazanılmasının mümkün olmadığını ifade eden sakınım özelliğinin sağlanması beklenir. Daha genel olarak, jetonların kaybedilmesi veya kazanılması modellenen DES'in fiziksel olarak sahip olduğu sakınım özelliğinin bir yansıması olmalıdır. Yani gerçek DES'te bu özellik varsa bu modelde de ortaya çıkmalıdır.

2.4.1.5 Canlılık

Açmaz olma ve kilitlenme özelliklerinin bir tamamlayıcısı "canlı" geçiş kavramıdır. Burada, herhangi bir geçişin mümkün olmaması ya da işaretli bir duruma geçişin mümkün olması kavramı yerine, verilen bir geçişin ateşlenme kabiliyeti ile ilgilenilecektir.

Tanım 2.11 "Canlılık":

x_0 ilk koşuluna sahip bir Petri ağı, x_0'dan herhangi bir duruma geçerken herhangi bir geçişin ateşlenme özelliğine sahip olan bir yörünge daima mevcut ise "canlı"dır denir.

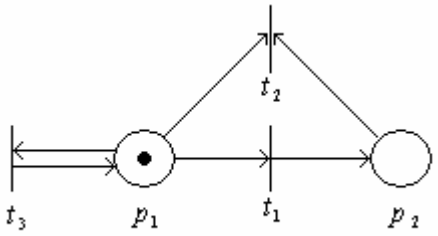
Ancak bu tanım çok katıdır ve test edilmesi de çok karışık olacaktır. O nedenle pratikte kullanılması pek mümkün değildir. Sadece 4 seviyeli bir canlılık sınıflandırmasına motivasyon olur.

x_0 ilk koşulu ile verilen Petri ağındaki bir geçiş,

- Ölü veya L0-canlıdır, eğer geçiş asla ateşlenmiyorsa
- L1-canlıdır, eğer x_0'dan başladıktan sonra en az bir kez ateşleniyorsa
- L2-canlıdır, eğer en az $k > 1$ kez ateşleniyorsa
- L3-canlıdır, eğer ∞ kez ateşleneceği bir dizi varsa
- Canlı veya L4-canlıdır, x_0'dan erişilen her duruma giden mümkün bütün yollar için L1-canlı ise.

Kapsanabilirlik kavramının L1-canlılık ile yakın ilişkisi vardır. Ölü geçişleri belirlemek için kapsanabilirlik testi yapmak mümkündür.

Örnek 2.6: (Canlılık)



Şekil 2.9 : Örnek 2.6 Petri Ağ Modeli

Şekil 2.9'da t_2 geçişi ölüdür çünkü asla ateşlenemez. t_1 geçişi L1-canlıdır çünkü bir kere ateşlenebilir. t_1 geçişi ateşlendiğinde yeni durumda tüm geçişler ölü olacaktır. t_3 geçişi L3-canlıdır çünkü ∞ kere ateşlenebilir, bu geçiş L4-canlıdır denemez çünkü t_1 geçişi ateşlendikten sonra artık ateşlenemez.

2.4.1.6 Kesintisiz Olma

Bazı durumda iki farklı geçiş aynı koşul kümesi ile izinli olabilir. Eğer biri ateşlenirse acaba diğerinin izinli olma özelliği aynı kalır mı? Genel olarak bunun garantisi yoktur. Gerçekte iki geçiş tam olarak aynı koşullara sahip olmazlar, ancak sadece bir koşul ortak olur. Kesintisiz olma özelliği, izinli bir geçişin başka bir izinli geçişin ateşlenmesi ile izinsiz hale geçmeme özelliğidir.

Tanım 2.12 "Kesintisiz Olma":

Eğer herhangi iki izinli geçiş için, birinin ateşlenmesi diğerinin iznini yok etmiyorsa, böyle bir Petri ağına "kesintisiz"dir denir.

Şekil 2.9'daki Petri ağı kesintisiz değildir çünkü t_1 ve t_3 izinlidir, ancak t_1 ateşlendiğinde t_3'ün izni kalkar. Diğer taraftan Şekil 2.5'teki kuyruk sistemi Petri ağı kesintisizdir.

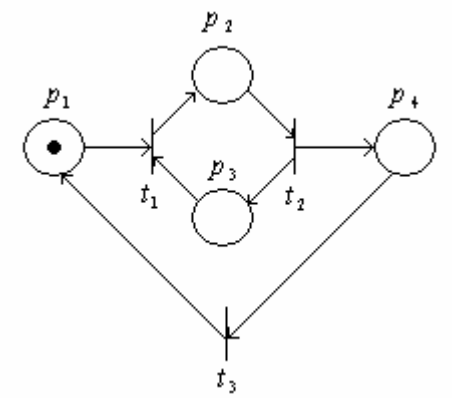
2.4.2 Lineer Cebirsel Teknikler

Erişilebilir durumlar ve sakınım gibi bazı problemleri çözmek için daha önce tanımlanan durum denklemleri kullanılabilir. Bu alternatif bir cebirsel teknik sağlar ve Petri ağlarının yapısal özelliklerini belirleme konusunda (ki bu yapısal özellikler Petri ağının topolojisinin bir sonucu A matrisinde ortaya çıkar) güçlü bir imkan sağlar. Bu durum denklemleri $x_{k+1} = x_k + u_k A$ olarak tanımlanmıştı. Erişilebilirlik açısından bu denkleme bakılırsa, bir x durumunun erişilebilir olmasının gerek koşulu seçilebilir. x_0 ilk koşulu için x durumuna erişilebilirliğine bakılmak istenirse,

$$vA = x - x_0 \qquad \textbf{(2.27)}$$

v'nin non-negatif olarak belirlenmesi gerekir. Eğer varsa, bu v'ye "ateşlemeleri sayan vektör" adı verilir. Bu v vektörü her geçişin kaç kere yapılması gerektiğini söyler. Bu sadece bir gerek koşuldur. v'nin non-negatif tamsayılar olarak varlığı bunların ateşleme izinlerinin olacağını göstermediği için, mümkün olduğunun garantisini vermez. Bu bir örnek üzerinde görülebilir.

Örnek 2.7:



Şekil 2.10 : Örnek 2.7 Petri Ağ Modeli

Şekil 2.10'deki Petri ağının A matrisi,

$A=\begin{bmatrix} -1 & 1 & -1 & 0 \\ 0 & -1 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$ ve ilk koşulu $x_0=[1,0,0,0]$'dır. $x=[0,0,0,1]$ durumu ele alınırsa,

$vA=x-x_0=[0,0,0,1]-[1,0,0,0]=[-1,0,0,1]$ denkleminden $v=[1,1,0]$ olarak elde edilir yani pozitif ateşleme dizisi var gibi görülür. Bunun anlamı t_1,t_2 veya t_2,t_1 dizisi uygulanınca elde edilir olmasıdır. Ancak bu iki ateşleme de x_0'da mümkün değildir. Bu, önerilen cebirsel tekniğin negatif yönüdür. Pozitif tarafı ise, $x'=[0,1,0,0]$'a x_0'dan erişilebilir mi diye araştırılmak istenirse, sistem denklemi,

$vA=x'-x_0=[0,1,0,0]-[1,0,0,0]=[-1,1,0,0]$ olur ve bu denklemin bir çözümü yoktur, x' erişilemezdir sonucuna varılır.

Yine $x_{k+1}=x_k+u_kA$ denklemi sakınım özelliği ile ilgili bazı bilgileri elde etmek için de kullanılabilir. A matrisine sahip bir Petri ağı ele alınsın. Eğer $A\gamma^T=0$ yapan $\gamma=[\gamma_1,\gamma_2,...,\gamma_n]$ $\gamma_i\geq 0$ $i=1,2,...,n$ bir γ var ise bu durumda şunlar yazılabilir.

$$x=x_0+vA \tag{2.28}$$

$$x\gamma^T=x_0\gamma^T+vA\gamma^T \tag{2.29}$$

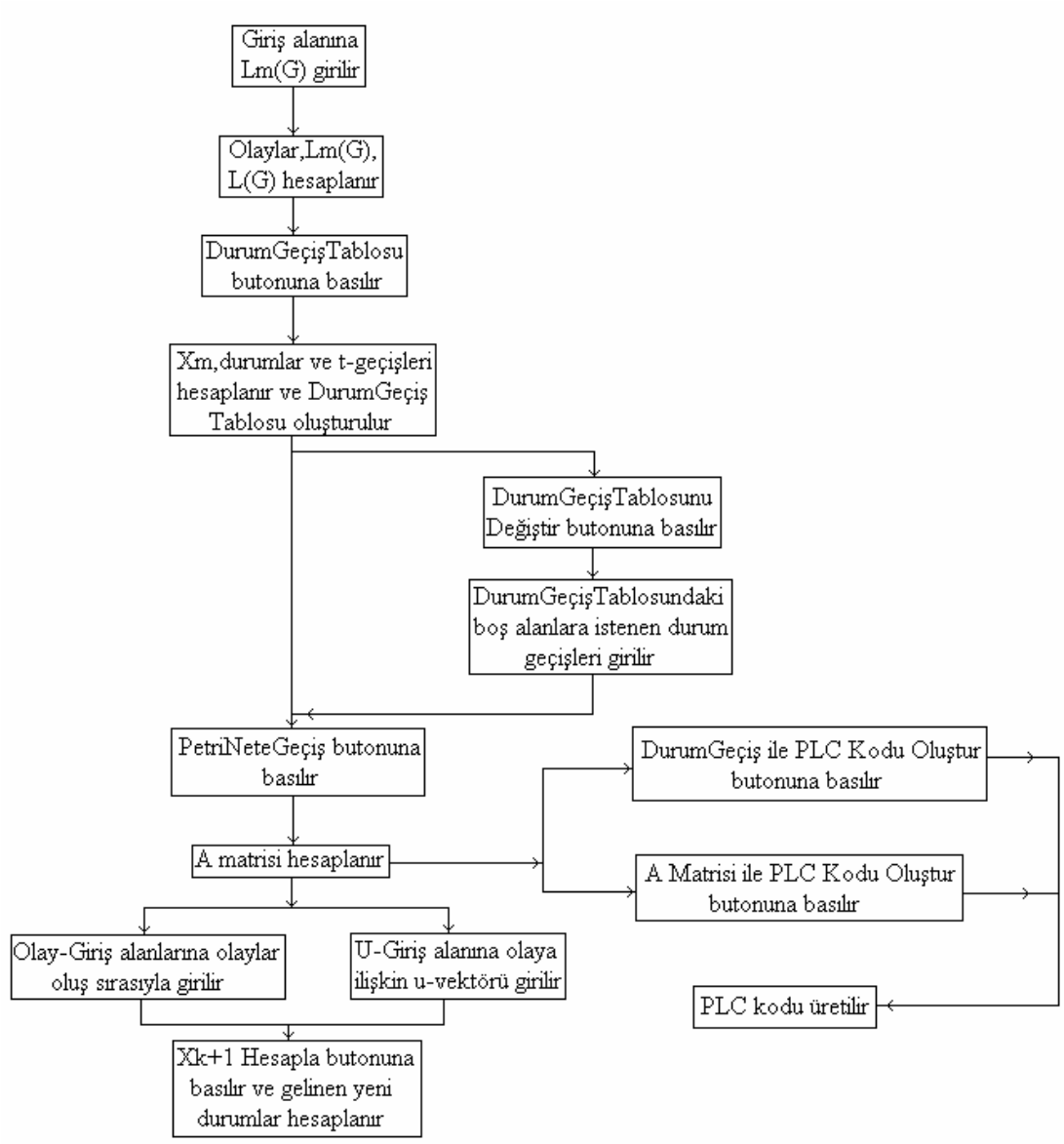
$$x\gamma^T=x_0\gamma^T \tag{2.30}$$

Son eşitlik, x_0'dan erişilebilir tüm x durumları için $x\gamma^T = st$ olan bir γ olduğunu söylediği için, bunun anlamı bu Petri ağının x_0'ın herhangi bir seçimi için "γ'ya göre sakınım" özelliğine sahip olduğudur.

3. PROGRAM AÇIKLAMALARI

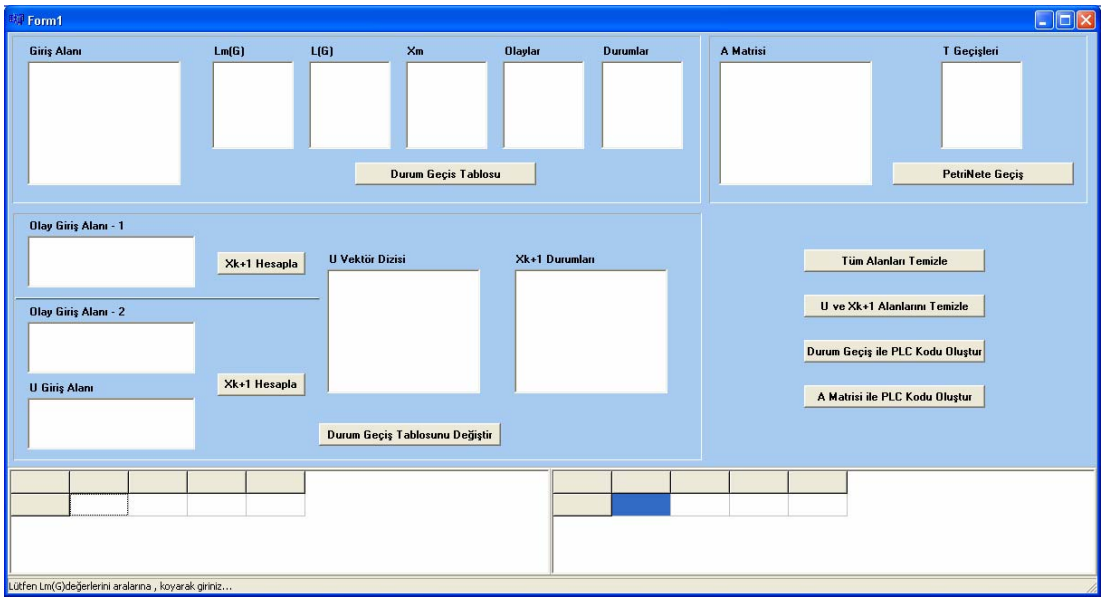
Bu çalışmada, biçimsel işaretli bir dilden endüstriyel bir işlemciye otomatik kod üreten bir program geliştirme amaçlanmıştır. Bu amaç için, önce işaretli dilden üretilen dile geçilmiş ve bu dili üreten otomatın durum geçiş diyagramı elde edilmiştir. Daha sonra durum geçiş matrisinden sistemin Petri ağ modeli elde edilmiş ve bu model temel alınarak kullanılan işlemciye uygun kod üreten yazılım gerçeklenmiştir. Bu işlemler, tüm aşamaları görsel olarak sunmaya imkan veren Borland C++ Builder ortamında gerçekleştirilmiştir. İzlenen yönteme ilişkin işaret akış diyagramı Şekil 3.1'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 yönteme ilişkin işaret akış diyagramı Şekil 3.3 ve Şekil 3.4 'de gösterildiği gibidir.



Şekil 3.1 : Programa İlişkin Akış Diyagramı

3.1 Borland C++ Builder Programı



Şekil 3.2 : Program Arayüzü

Program, "Giriş Alanı"na işaretli dilin (Lm(G)) girilmesiyle başlar. Lm(G)'ye ait kelimeler aralarına virgül 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 alanina 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 değişken olan string_len'de bir önceki
              kelimenin bittiği indeks tutulur. Böylece (string_len+1)
              den text_len'e kadar olan kısım yeni girilen kelimedir.
              Eğer yeni girilen kelimenin uzunluğu sıfır değilse ilgili
              tablolara gerekli eklemeler yapılır.
            */
            if((text_len != 0)&&((text_len-string_len)!=0))
             {
                 // Yeni girilen kelime LmTuple da saklanır.
                 LmTuple = GirisMemo->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. Eğer 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)
       {
            GirisMemo->Enabled = false;
       }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::GirisMemoKeyPress(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-geçişi, (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-geçişi]=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 XMListBox'a eklenir.
XmListBox->Items->Add("1");

/* Lm(G) deki kelimeler 'LmTuple' teker teker bulunur ve Kleen
  Kapaniş'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);

               //İşaretli 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 sira 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 Martisinde
           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 oluştuğu 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]=yeni_p≠ “ ” ise, p_t[j][i] ve t_p[j][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 sayısıyla 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 Xo=[1, 0 , .., 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 virgül konularak girilmesi sağlanır. İşlem bittiğinde ya ENTER tuşuna basılmalı veya virgül 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 Xk[j] = Xo[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 (Xk=Xo). Aksi halde belirlenen yeni durum "Xk", bir önceki durumdan "Xo", farklı ise sistem durum değiştirmiş demektir. Yeni durum Xo'a atanır (Xo=Xk). 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-geçişi durum 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 ulaştıysa başlangıç durumuna döndürülür.

```
//  U1AnilMemo alanina olaylar aralarina virgul 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 virgül konularak girilmesi sağlanır. İşlem bittiğinde ya ENTER tuşuna basılmalı veya virgül konulmalıdır. U1OzdeMemoKeyDown fonksiyonunda, girilen her “e” olayının DGStringGrid’deki sütun indeksi “e_indeks” bulunur. Sistemin durum değiştirip değiştirmediği belirlenir, DGStringGrid[e_indeks][Xo] = Xk ≠ “ ” ise sistem durum değiştirmiş demektir. Sistemi Xk. duruma geçiren t-geçişi 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 Xk, sistemin bulunduğu Xo durumuna atanır. Eğer sistem açmaz bir duruma ulaştıysa başlangıç durumuna dönülür.

  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 Xk[j] = Xo[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 (Xk=Xo). Aksi halde belirlenen yeni durum “Xk”, bir önceki durumdan “Xo”, farklı ise sistem durum değiştirmiş demektir. Yeni durum Xo’a atanır (Xo=Xk). Sistem durum değiştirsin ya da değiştirmesin yapılan her hesaplama için Xk durum vektörü “X_{k+1} Durumları” alanına eklenir. Eğer sistem açmaz bir duruma ulaştıysa başlangıç durumuna döndürülür.

```
// U1OzdeMemo alanına giriş yapılırsa aşağıdaki kontroller koşar.
// Olaylar tek tek ve aralarına virgül 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!","Uyari",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 karsılığı boş ise sistem
```

```
        // durum değiştirmiyor 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
        {
          // Eğer matristeki karşılığı boş değil ise hangi duruma (next_state) gidileceği bulunur.
          int next_state = StrToInt(DGStringGrid->Cells[col_no][previous_state]);

          // Aktif olan t geçişi (previous_state'ten next_state'e geçiren)
          // 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;
               }
          }

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

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

// Enter tuşuna basılınca U1OzdeMemo alanına giriş yapılmasına izin verilmez.
if(Key == VK_RETURN)
 {
      U1OzdeMemo->Enabled = false;
```

```
        }

}
//---------------------------------------------------------------------------

// Alttaki "Xk+1 Hesapla" tuşuna basıldı.
// UListBox'daki tüm U'lar için  Xk+1 = Xk + 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 Xk+1 = Xk + uA işlemine tabii
       // tutulur. u_index UListBox alanındaki U'larin 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 çarpmasında kullanılan UA, her u değişimimde 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];
               }
          }
```

```
        // Xk+1 = Xk + 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 dönülür.
       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 “U2VektorMemoEnter” 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. U2VektorMemoEnter 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 U2VektorMemo
// alanı sıfırlanır.
void __fastcall TForm1::U2VektorMemoEnter(TObject *Sender)
{
      U2VektorMemo->Text = "";
}
//---------------------------------------------------------------------------
// "U" alanina giriş yapıldığında aşağıdaki kontroller koşar.
void __fastcall TForm1::U2VektorMemoKeyPress(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 = U2VektorMemo->Text.Length();

      // İlk giriş boşluk olamaz ve üst üste boşluk girilemez.
      if(Key == VK_SPACE)
       {
            if((U2VektorMemo->Text == "")||(U2VektorMemo->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 Xk+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();
      AListBox->Items->Clear();
      XmListBox->Items->Clear();
      UListBox->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 kullanilan 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 edilecegi 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 Matrisininden 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 bağlı 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 Geciş Matrisi  \n";
Message20 = "DURUMGECISM: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 bağlı olmayan sabit kısımları PLC_Code_A.doc'a eklenir.
    fwrite(APLCMemo->Text.c_str(),APLCMemo->Text.Length(),1,fp);

    fclose(fp);

}
```

3.2 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. Gelinen 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.

3.2.1 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 durumundayken 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 gelinen 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 gelinen 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 Geciş Matrisi
DURUMGECISM: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.
// Gelinen 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;
```

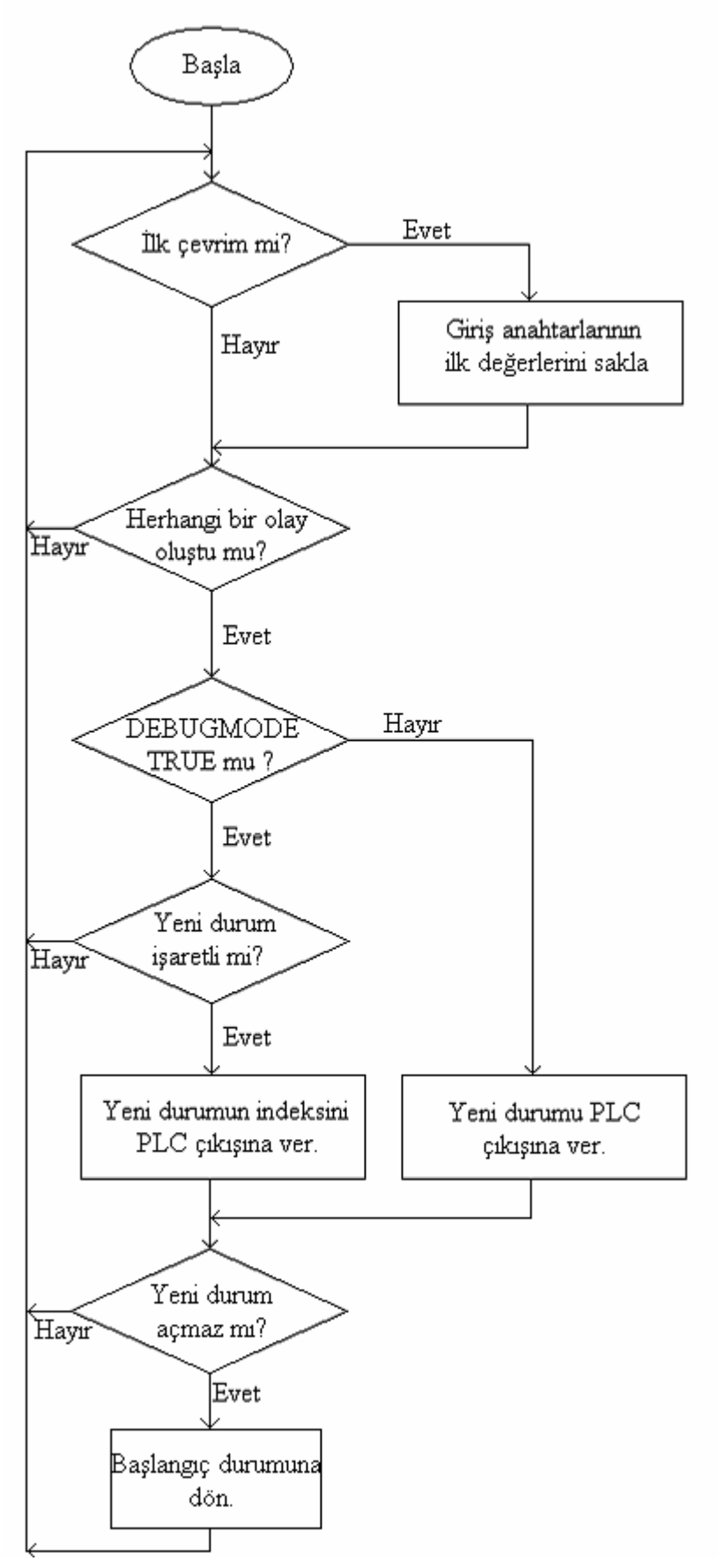
```
            Q[K,L]:= TRUE;

            // 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 3.3 : Durum Geçiş Diyagramı Yöntemi İşaret Akış Diyagramı

3.2.2 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 gelinen yeni durumdur. Oluşan olaya ait her t-geçişi için u girişleri belirlenir. Her durumda bir t-geçişi aktiftir yani durum değişimine sebep olur. Gelinen yeni durum X_0 dizisine atanır.

PLC çıkışları sıfırlanır ve DEBUGMODE değişkeninin değerine göre çıkışlar belirlenir. Programda gelinen 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 gelinen 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 Geciş Matrisi
DURUMGECISM: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,-1,0,0,1,0,0,0,0,0,-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 oldugu 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;

// Xk+1 = Xk + 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;
```

```
    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 değiştirmediği 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 Xo 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;
```

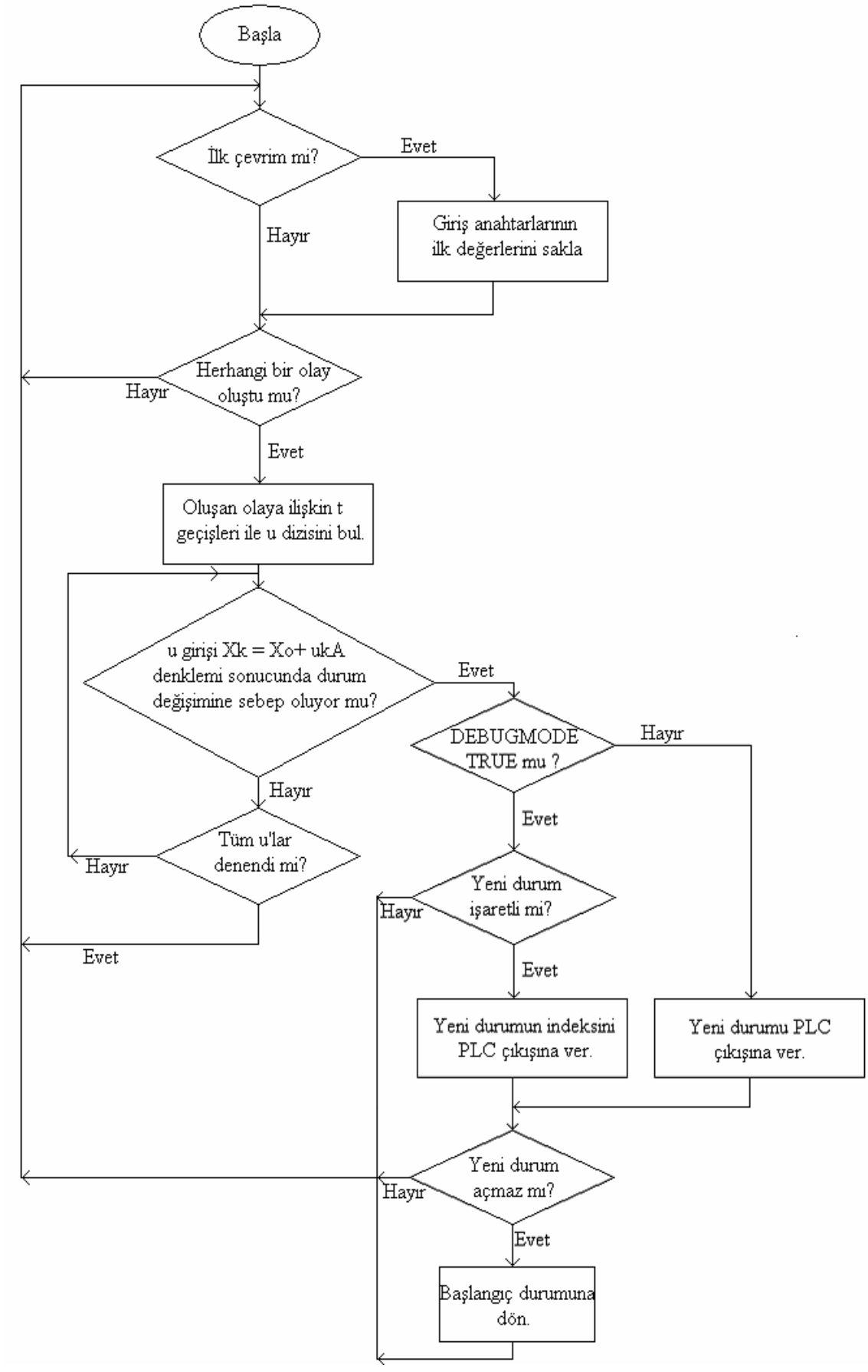
```
          K := I DIV 8;
          L := I MOD 8;
          Q[K,L]:= TRUE;

          // 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
```



Şekil 3.4 : Petri Ağı A Matrisi Yöntemi İşaret Akış Diyagramı

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 Yayincilik, Ankara.

ÖZGEÇMİŞ

Anıl Şahin 1978 yılında Çanakkale'de doğdu. Orta öğrenimini 1995 yılında Çanakkale Anadolu Lisesi'nde tamamladı. 1999 yılında İstanbul Teknik Üniversitesi'nde Kontrol ve Bilgisayar Mühendisi diplomasını aldı. 1999 yılında telekomünikasyon alanında faaliyet gösteren Nortel Netaş firmasında işe başladı. Halen bu firmada yazılım geliştirme mühendisi olarak çalışmaktadır.