İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AYRIK OLAY SİSTEMLERİNİN TASARIMI VE KONTROLÜ İÇİN YENİ BİR GERÇEKLEME VE OTOMATİK KOD ÜRETME YÖNTEMİ

DOKTORA TEZİ

İbrahim Tolga HASDEMİR

Anabilim Dalı : Elektrik Mühendisliği

Programı : Kontrol ve Otomasyon Mühendisliği

ŞUBAT 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AYRIK OLAY SİSTEMLERİNİN TASARIMI VE KONTROLÜ İÇİN YENİ BİR GERÇEKLEME VE OTOMATİK KOD ÜRETME YÖNTEMİ

DOKTORA TEZİ

İbrahim Tolga HASDEMİR
(504012022)

Tezin Enstitüye Verildiği Tarih : 15 Eylül 2008
Tezin Savunulduğu Tarih : 3 Şubat 2009

Tez Danışmanı : Doç. Dr. Salman KURTULAN (İTÜ)
Diğer Jüri Üyeleri : Doç. Dr. M. Turan SÖYLEMEZ (İ.T.Ü.)
Prof. Dr. Külmiz ÇEVİK (İ.T.Ü.)
Prof. Dr. Galip CANSEVER (Y.T.Ü.)
Prof. Dr. Coşkun SÖNMEZ (Y.T.Ü)

ŞUBAT 2009

ÖNSÖZ

Lisans öğrenciliğimden başlayarak yüksek lisans ve doktora öğrenimim boyunca her aşamada bilgi ve deneyimleri ile bana yol gösteren, sahip olduğum mühendislik ve akademik birikimde büyük payı olan, değerli hocam Doç. Dr. Salman KURTULAN'a, derslerinde, çalışma gruplarında ve sohbetlerinde akademik birikimi ve bilimsel bakışı ile desteğini hiçbir zaman esirgemeyen, yorum ve önerileri ile tez çalışmamda büyük katkıları olan değerli hocam Prof. Dr. Leyla GÖREN'e, değerli fikir, eleştiri ve yorumları için Yrd. Doç. Dr. Neslihan S. ŞENGÖR'e, gerek manevi gerekse akademik destekleri için değerli arkadaşlarım Araş. Gör. Dr. Özgür T. KAYMAKÇI ve Araş. Gör. İlker ÜSTOĞLU'na, sonsuz teşekkürlerimi sunarım.

Yaklaşık iki yıldır çalışmakta olduğum Siemens A.Ş. Otomasyon ve Kontrol Sistemleri Bölümü'nde anlayışları ve destekleri için başta Otomasyon Sistemleri Kısım Müdürü Sayın Sevtap İNAN, Eğitim Bölümü yöneticisi Sayın Murat S. ORAL ve değerli çalışma arkadaşım Elektronik Mühendisi Ergün ÇETİN olmak üzere tüm mesai arkadaşlarıma teşekkürü borç bilirim.

Son ve en özel olarak, sabır, özveri ve sevgileri ile ödenmesi mümkün olmayacak destekleri için aileme sonsuz teşekkürlerimi sunarım.

Eylül 2008 İbrahim Tolga HASDEMİR

İÇİNDEKİLER

Sayfa

ÖZET ..... **XV**
SUMMARY ..... **XVİİ**
1. GİRİŞ **1**
2. AYRIK OLAY SİSTEMLERİ **3**
2.1 Ayrık Olay Sistem Modelleri ve Dil kavramı 4
2.3 Dil ile İlgili Tanımlar 6
2.4 Dil Modeli İçin Otomat Gösterim Biçimi 7
2.4.1 Otomatın temsil ettiği diller 9
2.4.2 Kilitlenme 10
2.4.3 Deterministik olmayan otomat 11
2.5 Otomatlar Üzerine İşlemler 12
2.5.1 Tekli işlemler 12
2.5.1.1 Erişilebilir kısım 12
2.5.1.2 Kilitlenmesiz kısım 12
2.5.1.3 Budama 13
2.5.2 Çoklu işlemler 13
2.5.2.1 Çarpma 13
2.5.2.2 Paralel birleşim 14
2.6 Petri Ağları 16
2.7 Zamanlı Otomat 21
2.7.1 Saat yapısı 21
2.8 PLC-otomat 25
3.AYRIK OLAY SİSTEMLERİNİN KONTROLÜ **29**
3.1 Üstdenetim Kuramı 30
3.1.1 Kontroledilebilir alt ve üst diller 33
3.1.2 Kontrol davranışının tanımlanması 34
3.1.2.1 Yasaklı durumlar 35
3.1.2.2 Durum bölme 35
3.1.2.3 Olay dönüşümü 35
3.1.3 Üstdenetleyicilerin otomatlarla gerçeklenmesi 36
3.1.4 Modüler kontrol 37
3.1.5 Lokal modüler kontrol 39
3.1.6 Dağıtılmış kontrol 39
3.1.7 Hiyerarşik kontrol 40
3.2 Modele Dayalı Olmayan Yöntemler 41
3.3 Programlanabilir Lojik Kontrolör (PLC) 42
3.3.1 Gecikmeli kapatan zamanlayıcı (TON) 45
3.3.2 İleri sayıcı (CTU) 46
4. ZS-OTOMAT **47**
4.1 Zamanlama ve Sayma Davranışları 47
4.2 Zamanlama ve Sayma Davranışlarının İfadesi 50

4.3 ZS-otomat 55
4.4 ZS-otomat İçin Durum Diyagramı Gösterilimi 58
5. GERÇEKLEME 63
5.1 Standart Gerçekleme Yöntemi 64
5.1.1 Çığ etkisi sorunu 65
5.2 ZS-otomat İçin PLC ile Gerçekleme Yöntemi 66
5.2.1 Durum geçişlerinin ve ilk durumun gerçeklenmesi 67
5.2.2 ZS-yapısının ve çıkış fonksiyonların gerçeklenmesi 74
6. OTOMATİK PLC KOD ÜRETİMİ 77
6.1 OtoPLC: Otomatik PLC Kod Üretimi İçin Geliştirilen Matlab Uygulaması 77
6.1.1 Durum geçiş, çıkış fonksiyonu ve ZS-yapısının tanımlanması 77
6.1.2 Başlangıç değerlerinin atanması ve özçevrimlerin silinmesi 79
6.1.3 Geçişlere ilişkin durum ve olay indis kümelerinin atanması 80
6.1.4 Durum geçiş fonksiyonuna ilişkin Kurma ve Silme koşulları 81
6.1.5 PLC kodunun yazdırılması 82
7. SONUÇ VE ÖNERİLER 87
KAYNAKLAR 89
EK A. OTOPLC UYGULAMASI İLE ELDE EDİLEN PLC PROGRAMI 95
A.1. Ana Program 95
A.2. Alt Program 97

KISALTMALAR

DES	: Discrete event systems
AOS	: Ayrık olay sistemleri
PLC	: Programmable Logic Controller
STL	: Statement List
ST	: Structured Text
LAD	: Ladder Logic
FBD	: Functional Block Diagram
SFC	: Sequential Function Chart
TON	: On Delay Timer
TOF	: Off Delay Timer
CTU	: Counter Up
CTD	: Counter Down
CTUD	: Counter Up and Down
ZS-otomat	: Zamanlama ve Sayma yapısı olan otomat
ZY	: Zamanlama Yapısı
SY	: Sayma Yapısı

ÇİZELGE LİSTESİ

Sayfa

Çizelge 3.1 S7-200 PLC'ler için temel mantıksal işlem komutları 44
Çizelge 3.2 Merdiven mantığında değişkenler için kullanılan temel gösterimler ... 44
Çizelge 4.1 Durum diyagramı gösteriminde ZS-otomatın yapıtaşları 58

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1 : Zamana bağlı sürekli sitemlerle (a), ayrık olay sistemlerinin (b) durum uzayındaki davranışlarının karşılaştırılması 3
Şekil 2.2 : Örnek bir otomata ilişkin durum geçiş diyagramı 8
Şekil 2.3 : Örnek bir Petri ağı ve buna ilişkin iki farklı işaretleme (durum) 18
Şekil 2.4 : t_1 geçişinin gerçekleşmesinden sonra Şekil 2.3b'deki Petri ağına ilişkin yeni durum. 19
Şekil 2.5 : $\Sigma = \{\alpha\}$ olay kümeli bir AOS için olay-zaman diyagramı 21
Şekil 2.6 : $\Sigma = \{\alpha, \beta\}$, olayları sürekli aktif olan bir AOS için olay-zaman diyagramı 23
Şekil 2.7 : $\Sigma = \{\alpha, \beta\}$, α olayının her durumda aktif olmadığı olay-zaman diyagramı 24
Şekil 2.8 : Örnek bir PLC-otomata ilişkin durum diyagramı 27
Şekil 3.1 : Üstdenetleyicili kontrol yapısı 31
Şekil 3.2 : Olay dönüşümü için kullanılacak H_{ist} otomatı 35
Şekil 3.3 : İki üstdenetleyici ile oluşturulan bir modüler kontrol mimarisi 38
Şekil 3.4 : Dağıtılmış kontrol yapısı 40
Şekil 3.5 : PLC'de kullanıcı programının çevrimsel olarak yürütülmesi 42
Şekil 3.6 : $D = A \bullet \overline{B} + C$ ifadesini gerçekleyen LAD ve STL programları 44
Şekil 3.7 : TON zamanlayıcısının LAD ve STL dillerindeki kullanımı 45
Şekil 3.8 : Örnek TON zamanlayıcısına ilişkin zaman diyagramı 46
Şekil 3.9 : CTU sayıcısının LAD ve STL dillerindeki kullanımı 46
Şekil 3.10 : Örnek CTU sayıcısına ilişkin zaman diyagramı 46
Şekil 4.1 : σ olayının *n* kez sayılmasını modelleyen standart otomat gösterimi 51
Şekil 4.2 : σ_1 ve σ_2 olaylarının sayılmasını modelleyen standart otomat gösterimi 51
Şekil 4.3 : İki farklı olayın sayıldığı davranış için eşdeğer sembolik bir gösterim. 52
Şekil 4.4 : Sayma davranışına ilişkin eşdeğer otomat 53
Şekil 4.5 : Örnek 4.1'de tanımlanan ZS-otomat 59
Şekil 5.1 : (a) Örnek bir otomat, (b) bu otomata PLC karşılık gelen PLC programı 65
Şekil 5.2 : Çığ etkisi problemi bilinen yöntemlerle çözülemeyen basit bir otomat. 66
Şekil 5.3 : Eşitlik 5.9'a karşılık gelen merdiven dili programı. 70
Şekil 5.4 : Örnek 5.2' ye ilişkin PLC programı 71
Şekil 5.5 : Şekil 5.2' deki otomat için çığ etkisi problemi olmayan gerçekleme 72
Şekil 5.6 : İlk durum için PLC gerçeklemesi 74
Şekil 5.7 : ZS Yapısına ilişkin çıkış fonksiyonlarının gerçeklenmesi 75
Şekil 6.1 : Matlab uygulamasında örnek ZS-otomata ilişkin tanımlamalar. 79
Şekil 6.2 : İlk değerlerin atanması ve öz çevrimlerin silinmesi 80
Şekil 6.3 : I_{SQ}, $I_{T\Sigma}$ ve $I_{R\Sigma}$ indis kümelerinin oluşturulması 80

Şekil 6.4 : Kurma ve Silme koşullarının oluşturulması 81
Şekil 6.5 : Örnek ZS-otomat için "set_kosulu" ve "reset_kosulu" değişkenleri 82
Şekil 6.6 : OtoPLC programının ürettiği PLC kodunun örnek görüntüsü 83
Şekil 6.7 : Üretilen sembol tablosunun metin editöründeki görüntüsü..... 84
Şekil 6.8 : Sembol tablosunun MicroWin editöründeki görüntüsü 84
Şekil 6.9 : Ana ve alt PLC programlarının MicroWin editöründeki görüntüsü 85

SEMBOL LİSTESİ

$\mathbf{N}$	: Doğal sayılar kümesi
$\mathbf{N}^n$	: Elemanları doğal sayılar olan n boyutlu vektörler kümesi
$\mathbf{R}^+$	: Pozitif reel sayılar kümesi
$\mathbf{R}_{\geq 0}$	: Negatif olmayan reel sayılar kümesi
$\mathbf{2}^Q$	: Q kümesinin tüm altkümeleri
$\mathbf{\Sigma}^*$	: Σ olay kümesinin Kleene kapanışı
$\mathbf{L}^*$	: L dilinin Kleene kapanışı
$\boldsymbol{\varepsilon}$	: Uzunluğu sıfır olan (boş) kelime
$\bar{\mathbf{s}}$	: s kelimesinin tüm önekleri
$\bar{\mathbf{L}}$	: L dilinin tüm önekleri
$\mathbf{L(G)}$	: G otomatının ürettiği dil
$\mathbf{L_m(G)}$	: G otomatının işaretlediği dil
$\mathbf{Ac(G)}$	: G otomatının erişilebilir kısmı
$\mathbf{CoAc(G)}$	: G otomatının kilitlenmesiz kısmı
$\mathbf{Trim(G)}$	: Budanmış G otomatı
$\mathbf{G_1 \times G_2}$	: G_1 ve G_2 otomatlarının çarpımı
$\mathbf{G_1 \,\|\| \, G_2}$	: G_1 ve G_2 otomatlarının paralel birleşimi
$\mathbf{L_1 \,\|\| \, L_2}$	: L_1 ve L_2 dilleri için paralel birleşim işlemi
$\mathbf{P_i(L)}$	: L dilinin Σ_i olay kümesine izdüşümü
$\mathbf{P_i^{-1}(L)}$	: L dilinin Σ_i olay kümesinden ters izdüşümü
$\mathbf{S/G}$	: S üstdenetleyicisi kontrolündeki G ayrık olay sistemi
$\mathbf{L(S/G)}$	: S üstdenetleyicisi kontrolündeki G ayrık olay sisteminin ürettiği dil
$\mathbf{L_m(S/G)}$	: S üstdenetleyicisi kontrolündeki G ayrık olay sisteminin işaretlediği dil
$\mathbf{\Sigma_{uc}}$	: Kontrol edilemeyen olaylar kümesi
$\mathbf{K}^{\uparrow C}$	: Kontroledilebilir alt dillerin supremumu
$\mathbf{K}^{\downarrow C}$	: Önek kapalı kontroledilebilir üst dillerin infimumu
$\mathbf{AG(u)}$	: **u** vektörünün ağırlığı
$\mathbf{AG_{\downarrow}(u^k)}$	: $\mathbf{u}^k$ (k=1,2,..) vektör dizisindeki ağırlık kayıp elemanı
$\boldsymbol{\delta}\mathbf{(q)}$	: ZS-otomatta duruma bağlı çıkış fonksiyonu
$\boldsymbol{\delta}^T\mathbf{(q)}$	: ZS-otomatta duruma ve zamanlama yapısına bağlı çıkış fonksiyonu
$\boldsymbol{\delta}^C\mathbf{(q)}$	: ZS-otomatta duruma ve sayma yapısına bağlı çıkış fonksiyonu
$\mathbf{n}_q^k$	: Sayma vektörünün k adımındaki değeri
$\boldsymbol{\sigma}^T$	: Zamanlama olayı
$\boldsymbol{\sigma}_i^C$	: σ_i olayına karşılık gelen sayma olayı

$\mathbf{I_Q}$: Durum indis kümesi

$\mathbf{I_\Sigma}$: Olay indis kümesi

$\mathbf{I_{SQ}(i)}$: q_i durumuna geçişe ilişkin durum indis kümesi

$\mathbf{I_{S\Sigma}(i)}$: q_i durumuna geçişe neden olan olay indis kümesi

$\mathbf{I_{T\Sigma}(i,j)}$: q_j durumundan q_i duruma geçişe neden olan olay indis kümesi

$\mathbf{I_{R\Sigma}(i)}$: q_i durumundan çıkışa neden olan olay indis kümesi

AYRIK OLAY SİSTEMLERİNİN TASARIMI VE KONTROLÜ İÇİN YENİ BİR GERÇEKLEME VE OTOMATİK KOD ÜRETME YÖNTEMİ

ÖZET

Otomatlar, ayrık olay sistemlerinin analiz, tasarım ve kontrolü için formal yöntemlerin uygulanmasında durum tabanlı modelleme biçimi olarak yaygın şekilde kullanılmaktadır. Ayrık olay sistemlerinin (AOS) geribeslemeli kontrolü için kuramsal bir yapı tanımlayan Üstdenetim Kuramı ve kurama ilişkin uygulamalar, otomatların kullanım alanlarına örnek olarak verilebilir. Otomatın standart tanımının tasarıma ilişkin ayrık olay sistem davranışını tanımlayabilmesine rağmen, zamanlama ve sayma gibi bazı davranışların ifadesi bu modelleme biçimi ile kolay değildir.

Bu tezin temel amacı, AOS davranışlarının, özel olarak kontrol kurallarının tasarımını ve ifadesini mümkün kılan bir modelleme biçiminin geliştirilmesi ve bu modele dayanarak tasarlanmış davranışın uygulanabilmesi için bir yöntem elde edilmesidir. ZS-otomat olarak anılan yeni modelleme biçimi, tasarımcının, zamanlama ve sayma davranışlarını durum tabanlı gösterime atamasını mümkün kılan Zamanlama ve Sayma Yapısına sahiptir. Model, gerçekleme aşamasında kullanılan fiziksel zeminin sağladığı araçları kullanabilecek şekilde tanımlanmaktadır. Tezde, modelleme biçiminin yanı sıra, bir gerçekleme yöntemi de tanıtılmaktadır. Gerçekleme yöntemi tasarlanmış kontrolörün ya da üstdenetleyicinin Programlanabilir Lojik Kontrolörlerle (PLC) gerçeklenmesi için gerekli adımları sistematik olarak tanımlamaktadır. Yöntemin, tasarım davranışını yanlış olarak gerçeklemeye neden olan "çığ etkisi" adlı bir problem için çözüm oluşturduğu da gösterilmiştir. Elle uygulanabilirliğinin kolay olmasının yanında, bu yöntem PLC kodlarını otomatik olarak üretmek üzere programlanabilir olma özelliğine de sahiptir. Bu özellik, kontrol davranışını ifade eden ZS-otomata karşılık gelecek PLC programını üreten bir Matlab uygulamasının geliştirilmesinde kullanılmıştır.

A NEW REALIZATION AND AUTOMATIC CODE GENERATION METHOD FOR THE DESIGN AND CONTROL OF DISCRETE EVENT SYSTEMS

SUMMARY

When employing formal methods for the analysis, design and control of discrete event systems automata are extensively used as state based formalisms. Supervisory Control Theory which defines a theoretical framework for the feedback control of discrete event systems (DES) and its applications can be given as an example for the use of automata. Though the standard definition of the automaton is capable of describing the DES behavior of a design, some behaviors like timing and counting can not directly or easily be represented in the standard formalism.

The main objective of this thesis is to develop a formalism that enables designing and expressing DES behaviors, particularly control strategies, and to obtain a methodology in order to implement the designed behavior based on the introduced formalism. The new formalism, named TC-automaton, has a so called Timing and Counting Structure that enables the designer to assign timing and counting behaviors to the state based representation. It is defined in such a way that, in the realization stage, it is possible to make use of the tools provided by the physical realization platform. An implementation methodology is also introduced for the new formalism. The methodology systematically defines the steps for realizing the designed controller or supervisor via Programmable Logic Controllers (PLC). It is also shown that, the methodology resolves "avalanche effect" problem, which results with incorrect realizations of the design. Besides being straightforward in terms of manual applicability, the methodology can also be programmed to generate PLC codes automatically. This feature has been used to develop a Matlab application that generates a PLC program corresponding to the TC-automaton that expresses the control behavior.

1. GİRİŞ

Bilgisayar ve haberleşme teknolojilerindeki hızlı gelişmeler günümüzde yeni bir dinamik sistem türünü ortaya çıkarmıştır. Karakteristik olarak çoğunlukla teknolojik ve karmaşıklığı yüksek olan bu tür sistemlere farklı alanlardan çok sayıda örnek vermek mümkündür. Endüstriyel kumanda sistemleri, bilgisayar ve haberleşme ağları, otomobillerde ya da bina teknolojilerinde kullanılan gelişmiş izleme ve kontrol sistemleri, hava trafik kontrol sistemleri, demiryolu sinyalizasyon ve anklaşman sistemleri gibi birçok alanda bu tür sistemlerin varlığından bahsedilebilir. Bu tür sistemlerin davranışının ifadesi zaman içerisinde anlık olarak gerçekleşen *olay*lara bağlı olarak verilebilmektedir. Bunlar bir makinenin çalıştırılması, bir butona basılması, trafik ışığının kırmızıya geçmesi, bir veri paketinin gönderilmesi, ya da çalışan bir makinenin arızaya geçmesi gibi anlık olaylar olabilir. Anlık olayların meydana gelmesi sistemi bir durumdan başka bir duruma taşımaktadır. Bu şekilde, bu tür sistemlerin dinamiği, zamanda eşzamansız (asenkron) ve anlık olarak oluşan olaylara bağlı belirlenmektedir ve bu özellikleri ile *Ayrık Olay Sistemleri-AOS* (discrete event systems-DES) şeklinde adlandırılan bir dinamik sistem sınıfını oluşturmaktadırlar.

Günümüzde ayrık olay sistemleri olarak sınıflandırılabilecek dinamik sistemlerin yaygınlığı ve karmaşıklığı bu sistemlerle ilgili analiz, tasarım, kontrol ve optimizasyon gibi uygulamalarda sistem ve kontrol teorisindekine benzer sistematik kuralların ve yöntemlerin tanımlanmasını gerektirmektedir. Literatürde ayrık olay sistemleri ile ilgili akademik çalışmalar oldukça yaygındır. Özellikle, ayrık olay sistemlerinin kontrolüne yönelik *Üstdenetim Kuramı* son 20 yıl içinde akademik dünyada büyük ilgi uyandırmış, kuramın ortaya atılmasından [1] günümüze, kurama katkıda bulunan birçok çalışma yayınlanmıştır [2-15]. Çalışmaların bir bölümü gerçeklemeye yöneliktir. *Gerçekleme* ile kastedilen, bir ayrık olay sisteminin kontrolü için yapılan soyut tasarımla aynı davranışı gerçek dünyada sergileyecek bir uygulamanın teknolojik araçlar kullanılarak geliştirilmesidir. Bu nedenle tasarımı gerçekleyecek yöntemlerin, kısaca gerçekleme yöntemlerinin tanımlanması

gerekmektedir. Ancak, iyi tanımlanmış bir gerçekleme yönteminin varlığı altında bile gerçeklemenin tam ve sorunsuz olacağını söylemek kolay değildir. Tasarımın soyut ifadesinin varsayımları ile gerçeklemede kullanılan fiziksel ortamın kurallarının örtüşmemesi, tasarımın tam ve doğru bir karşılığının elde edilmesini zorlaştırabilmektedir. Bu nedenle ayrık olay sistemlerinin kontrolünde gerçekleme önemli bir çalışma konusudur ve bu konuda araştırmalar sürmektedir [17-35]. Araştırmalarda çoğunlukla kuramsal tasarımın teknolojik araçlarla hayata geçirilmesine yönelik uygulamalar [17-19,23,31,32], yöntemler [20,22,24-30,33-35], ya da uygulamada ortaya çıkan sorunlar [19,22-26,33-35] tanımlanmaktadır. Kuramsal ifadenin teknolojik araçlarda tam bir karşılığının bulunmaması gerçekleme için kesin ve sistematik bir çözümün verilmesini zorlaştıran temel sebeptir. Sistematik bir gerçekleme yönteminin varlığı, yöntemin programlanabilir olmasını, yani tasarımı teknolojik araçlarda (örneğin PC ya da PLC) gerçekleyen program kodunun da otomatik olarak üretilmesini mümkün kılacaktır. Bu çalışmanın katkısı, gerçekleme için yeni bir ayrık olay sistem gösterim modelinin geliştirilmesi, gerçekleme için kesin ve sistematik bir yöntemin tanımlanması ve bu yöntemden hareketle otomatik kod üretimini mümkün kılan bir Matlab uygulamasının geliştirilmesi şeklinde olacaktır. Bu yöndeki ilk sonuçlar için [33-35] numaralı kaynaklar görülebilir.

Bölüm 2'de ayrık olay sistemlerinin ifadesi için sitemin durum uzayındaki davranışını tanımlayan farklı model yapıları tanıtılacaktır. Bölüm 3'te ayrık olay sistemlerinin kontrolüne yönelik Üstdenetleyici Kuramı ve kontrol tasarımının gerçeklenmesinde kullanılan yöntemler, yaklaşımlar ve araçlar tanıtılacaktır. Bölüm 4'te ayrık olay sistemlerinin özellikle kontrolü ve gerçeklenmesine yönelik durum diyagramlarını temel alan yeni bir gösterim biçimi, Bölüm 5'te ise bu modele ilişkin bir gerçekleme yöntemi tanıtılacaktır. Bu gerçekleme yöntemi, Bölüm 6'da Programlanabilir Lojik Kontrolörlerde otomatik kod üretimi sağlayan bir Matlab uygulamasının geliştirilmesi için kullanılacaktır.

2. AYRIK OLAY SİSTEMLERİ

Sistem ve kontrol teorisi için analiz ve tasarımda çıkış noktası diferansiyel denklemler ya da fark denklemleridir. Burada serbest değişken zamandır ve sistem dinamiği zamana bağlı olarak ifade edilir. Mekanik, elektrik, elektromekanik ya da ısıl sistemler gibi fiziksel sistemler fizik yasaları ile belirlenen diferansiyel denklem takımları ile ifade edilir. Bu tür sistemlerin analiz, tasarım ve optimizasyonunda yer değiştirme, hız, ivme, akım, gerilim, basınç, sıcaklık gibi büyüklüklerle ilgilenilir. Sistem ve kontrol teorisinde bu büyüklükler, zaman *sürekli* olarak ilerlerken reel sayı değerleri alan *sürekli değişkenler*dir. Sürekli sistemlerde, X durum uzayı, **R** reel sayılar kümesi olmak üzere x(t) durumu bu kümeden

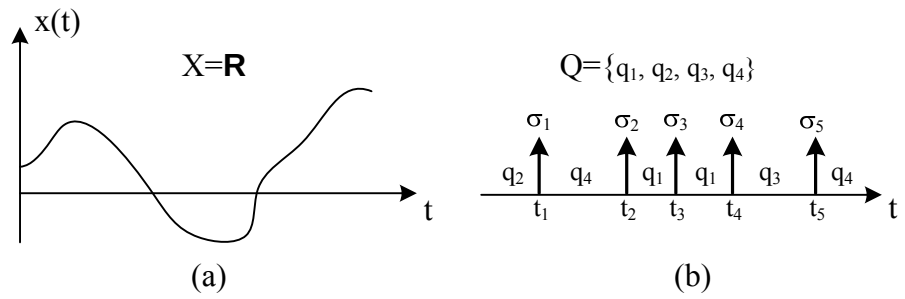
$$\dot{x}(t) = f(x(t), u(t), t) \qquad \textbf{(2.1)}$$

ifadesi ile belirlenen herhangi bir değer alabilir. Bunun ayrık zamandaki karşılığı

$$x(k+1) = f(x(k), u(k), k) \qquad \textbf{(2.2)}$$

şeklinde bir fark denklemidir. Burada u, sisteme uygulanan giriştir.

Ayrık olay sistemlerinde ise durum uzayı $Q=\{q_1, q_2, ..., q_n\}$ gibi ayrık bir kümedir ve bir olay meydana geldiğinde sistem ayrık bir durumdan bir diğerine geçer. Sistemin durum geçişi zamana bağlı olarak değil, herhangi bir anda oluşabilen olaylarla belirlenmektedir. Şekil 2.1'de zamana bağlı sürekli bir sistem ile durum uzayı $Q = \{q_1, q_2, q_3, q_4\}$ olan bir ayrık olay sisteminin zamana ve olaylara bağlı durum yörüngelerini karşılaştıran bir örnek görülmektedir.



Şekil 2.1 : Zamana bağlı sürekli sitemlerle (a), ayrık olay sistemlerinin (b) durum uzayındaki davranışlarının karşılaştırılması

Şekil 2.1b'de görüldüğü gibi ayrık olay sisteminin durumu bir olay meydana geldiğinde değişmektedir, ancak olayın meydana gelmesi σ_3 olayında olduğu gibi her zaman bir durum değişikliği olacağı anlamında gelmemektedir.

Görüldüğü üzere ayrık olay sistemlerinin davranışı için $\dot{x}(t) = f(x(t), u(t), t)$ benzeri zamana bağlı bir ifade verilmesi mümkün değildir. Sistemin durum uzayındaki yeri sadece olaylarla belirlenmektedir.

2.1 Ayrık Olay Sistem Modelleri ve Dil kavramı

Şekil 2.1b' de dört durumlu örnek ayrık olay sistemi için verilen durum yörüngesi aşağıdaki gibi olayların olma zamanları ile beraber bir olay dizisi şeklinde verilebilir.

$$(\sigma_1, t_1), (\sigma_2, t_2), (\sigma_3, t_3), (\sigma_4, t_4), (\sigma_5, t_5) \qquad \textbf{(2.3)}$$

Bir ayrık olay sisteminin (2.3)'te verildiği gibi oluşturabileceği tüm olay dizileri kümesi ele alınsın. Bu küme, sistem için *zamanlı dil* modelini oluşturmaktadır. Modelin "dil" olarak adlandırılmasının nedeni sisteme ilişkin olayların oluşturduğu $\Sigma = \{\sigma_1, ... \sigma_5\}$ kümesinin bir alfabe, bu kümenin elemanlarının harf, bu harflerden oluşan olay dizilerinin ise kelime şeklinde değerlendirilebilmesidir. Sistem için geçerli olan tüm olay dizilerinin oluşturduğu küme ise Σ alfabesi ile oluşturulmuş dil olacaktır.

Ayrık olay sistemleri için ele alınan olay dizileri ile beraber olaylara ilişkin zaman bilgisinin taşınıp taşınmadığı ya da nasıl taşındığına bağlı olarak üç farklı türden dil modelinden bahsedilebilir. Bir ayrık olay sisteminin modeli oluşturulurken en genel durumda hangi olayın tam olarak hangi anda oluştuğu bilgisi bulunmayabilir. Ancak olaylar arasındaki süreler için olasılık dağılım fonksiyonu biliniyorsa olayların olma zamanları için stokastik bir model verilebilir. Böylece stokastik dil modelinde ayrık olay sistemi için mümkün olan olay dizilimleri istatistiksel bilgileri ile beraber verilmiş olur. İstatistiksel bilgiler çıkarılırsa elde edilen zaman bilgileri ile beraber sistem için geçerli olan tüm olay dizileri yani zamanlı dil modeli elde edilir. Son olarak zaman bilgileri de çıkarılırsa sistem için mümkün olan tüm olay dizilimleri, yani zamansız dil modeli ya da kısaca dil modeli elde edilmiş olur [36,37].

Yukarıda belirtildiği gibi ayrık olay sistemleri için dil, zamanlı dil ve stokastik zamanlı dil olmak üzere üç farklı dil modeli söz konusudur. Hangi dil modelinin kullanılacağı tamamen yapılacak analizin amaçları ile ilgilidir. Birçok durumda ele alınan ayrık olay sitemi için mümkün olan olay dizileri ya da sistemin durum yörüngesinin ne olduğu ile ilgilenilir. Zamansız bir dil modeli böyle bir analiz için yeterli olacaktır. Ancak sistem davranışında sistemin belirli durumda ne kadar kaldığı ya da belirli bir duruma ne kadar sürede gelineceği gibi zamana ilişkin bilgilerle de ilgileniliyorsa zamanlı bir dil modelinin kullanılması gerekecektir. Ayrık olay sistemlerinin modellenmesinde olaylara ilişkin zaman bilgileri kesin olarak bilinmiyor, ancak olayların zamanlarına ilişkin istatistiksel bilgiler biliniyorsa sistemin stokastik zamanlı dil modeli üzerinden analiz ve tasarımı yapılabilir.

Bu çalışmada temel olarak zamansız dil modelleri ile ilgilenilecektir. Ancak, Bölüm 4'te tanıtılacak tasarım yöntemi için zamanlı dil modeli de irdelenecektir. Bu nedenle bu bölümde ilerleyen başlıklarda zamanlı ve zamansız dil modelleri üzerinde durulacaktır.

Görüldüğü gibi ayrık olay sistemlerinin dil modelleri olay dizilimlerini zamanlı ya da zamansız olarak verilmesine dayanmaktadır. Bu türden bir modelleme ayrık olay sisteminin tüm davranışlarının ifadesi için yeterli olmakla birlikte bu haliyle analiz ve tasarım için kullanılmaya elverişli değildir. Örneğin, çoğu durumda ele alınan ayrık olay sistemine ilişkin dil sonlu olmamaktadır, yani olası tüm olay dizilimlerini listeleyerek dilin ifade edilmesi mümkün değildir. Bu nedenle ayrık olay sistemlerinin davranışlarının ifadesi için sonlu yapılara gereksinim vardır. Sistem davranışını temsil eden bu yapılar *modelleme biçimleri* ya da *gösterim biçimleri* olarak adlandırılır. İlerleyen bölümlerde görüleceği gibi ayrık olay sistem davranışı ve sistemin özelliklerini yapılarında bulunduran bu gösterim biçimleri analiz ya da kontrolör tasarımı gibi uygulamalarda doğrudan kullanılabilecektir. Bu çalışmada tanımı ileride verilecek olan *otomat* gösterim biçimi temel alınacaktır. Literatürde sıklıkla karşılaşılan bir diğer modelleme biçimi olan Petri ağları da kısaca tanıtılacaktır.

Aşağıda buraya kadar bahsedilen kavramlara ilişkin formal tanımlara yer verilmiştir.

2.3 Dil ile İlgili Tanımlar

Verilen bir ayrık olay sisteminin olay kümesi Σ sistemin diline ilişkin alfabe olarak değerlendirilebilir. Bu kümenin sonlu olduğu varsayılacaktır. Bu kümedeki elemanlarla elde edilen olay dizeleri *kelime* olarak adlandırılır. Bir kelimedeki olay sayısına bu kelimenin *uzunluğu* denir. Uzunluğu sıfır olan kelimeye *boş kelime* denir ve ε ile gösterilir [36].

Bir kelime olayların ya da farklı kelimelerin eklenmesi ile oluşturulabilir. Kelimeler ya da diller için ekleme işlemi tanımlıdır. Örneğin $\Sigma = \{a, b\}$ olay kümesi üzerinde tanımlı abb şeklindeki bir kelime ab kelimesinin b olayı ile (ya da uzunluğu bir olan b kelimesiyle) eklenmesiyle oluşturulabilir. ab kelimesi ise a ve b' nin eklenmesiyle oluşur. Boş kelime ε ekleme işleminin etkisiz elemanıdır: u bir kelime olmak üzere $\varepsilon u = u\varepsilon = u$ eşitliği sağlanır.

Bir s kelimesine ilişkin t, u, $v \in \Sigma^*$ ve tuv = s olmak üzere t, s'nin bir *öneki*, u *altkelimesi* ve v *soneki* olarak adlandırılır.

Kelimeler ile ilgili bu temel tanımlardan sonra diller üzerinde işlemler için formal tanımlar aşağıdaki gibi verilebilir.

Tanım 2.1 (Dil) Bir Σ olay kümesi üzerinde tanımlı sonlu uzunluklu olay dizilerinden oluşan kümeye dil denir.

Örneğin $\Sigma = \{a, b, c\}$ için $L_1 = \{\varepsilon, a, ab, abc, abbc\}$, Σ olay kümesi üzerinde tanımlı beş kelimeden oluşan bir dildir.

Tanım 2.2 (Kleene-kapanışı) Bir Σ olay kümesi üzerinde Σ'nın oluşturacağı bütün sonlu kelimeler kümesine Kleene-kapanışı denir ve Σ^* ile gösterilir.

Örneğin $\Sigma = \{a, b, c\}$ için $\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \ldots\}$ ' dir.

Dolayısı ile bir Σ olay kümesi üzerinde tanımlı herhangi bir dil Σ^* ' ın bir alt kümesi olacaktır.

Kleen kapanışı tanımı dillere genişletilerek dil üzerine bir işlem olarak tanımlanabilir: $L \subseteq \Sigma^*$ olmak üzere L dilinin Kleene Kapanışı $L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup LLLL \ldots$ ile ifade edilir. * işlemi idempotenttir, yani $(L^*)^* = L^*$ ilişkisi sağlanır.

Tanım 2.3 (Dillerin ekleme işlemi) $L_a, L_b \subseteq \Sigma^*$ olsun. Bu durumda L_a dilinin L_b diline olan eki $L_a L_b = \{s \in \Sigma^* : (s = s_a s_b) \text{ve} (s_a \in L_a) \text{ve} (s \in L_b)\}$ şeklinde tanımlanır. Yani bir kelime L_a dilindeki bir kelime ile L_b dilindeki bir kelimenin eklenmesi ile oluşturulabiliyorsa bu kelime $L_a L_b$ dili içindedir.

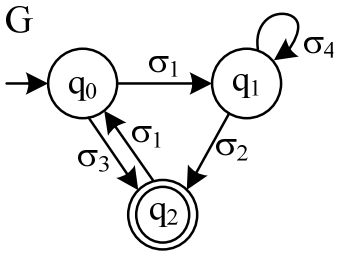
Tanım 2.4 (Bir dilin Önek-kapanışı) $L \subseteq \Sigma^*$ olmak üzere L dilinin önek-kapanışı $\overline{L} = \{s \in \Sigma^* : \exists t \in \Sigma^* (st \in L)\}$ şeklinde tanımlanır. Bir dilin önek kapanışı bu dilin tüm kelimelerinin tüm öneklerinden oluşmaktadır. Bir L dili ile bu dilin öneki $\overline{L}$ arasında $L \subseteq \overline{L}$ ilişkisi sağlanır.

2.4 Dil Modeli İçin Otomat Gösterim Biçimi

Ayrık olay sisteminin davranışının ifadesinde dil modeli, sisteme ilişkin olası tüm olay dizilimlerinin bir küme olarak oluşturulmasına dayanmaktadır. Bu haliyle dil modeli analiz, tasarım ve kontrol gibi uygulamalarda kullanılabilir değildir. Çoğu uygulamada sisteme ilişkin dil sonsuz bir kümeye karşılık gelmektedir. Bu nedenle daha önce de belirtildiği gibi, dili sonlu bir yapıyla ifade edecek gösterim biçimlerine gereksinim duyulur. Otomat ise bu amaçla sıklıkla kullanılan bir gösterim biçimidir. Aşağıda öncelikle otomat gösterimi durum geçiş diyagramları ile tanıtılacak, daha sonra formal tanımı verilecektir. Otomatlar ile dil arasındaki ilişkiye ise ilerleyen paragraflarda değinilecektir.

Otomat gösterimi en basit şekilde durum geçiş diyagramları ile ifade edilebilir. Durum geçiş diyagramlarında daireler durumlara, yönlü okların etiketi olaylara karşılık gelir. Bir okla işaret edilen durum ilk durumdur. Yönlü oklar, otomata ilişkin *durum geçiş fonksiyonları*nı tanımlamaktadır. Şekil 2.2' de örnek bir otomata ilişkin durum geçiş diyagramı görülmektedir. Bu G otomatının durum kümesi $Q = \{q_0, q_1, q_2\}$, olay kümesi $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$' dir. Şekilden de görüldüğü gibi otomatın bazı durumları iç içe çift daire ile gösterilmektedir. Bu durumlar *işaretli durum* olarak adlandırılır ve ileride bahsedileceği gibi otomatın modellediği davranışa ilişkin tamamlanan bir işleve karşılık gelmektedir. Bu örnek için işaretli durum kümesi $Q_m = \{q_2\}$ şeklindedir. Bu G otomatı ile tanımlanan durum geçiş fonksiyonu: $f(q_0, \sigma_1) = q_1$, $f(q_0, \sigma_3) = q_2$, $f(q_1, \sigma_4) = q_1$, $f(q_1, \sigma_2) = q_2$, $f(q_2, \sigma_1) = q_0$ şeklindedir. $f(q_0, \sigma_1) = q_1$ ile q_0 durumunda σ_1 olayı olursa otomatın q_1 durumuna

geçeceği tanımlanır. $f(q_1,\sigma_4)=q_1$ ile görüldüğü gibi q_1 durumda σ_4 olayının olması bir durum değişikliğine neden olmamaktadır. Bu, Şekil 2.2' de görüldüğü gibi durum geçiş diyagramında aynı duruma geri dönen bir okla gösterilir ve *özçevrim* olarak adlandırılır.



Şekil 2.2 : Örnek bir otomata ilişkin durum geçiş diyagramı

Otomatla ilgili bu temel bilgilerden sonra otomatın formal tanımı aşağıdaki gibi verilebilir.

Tanım 2.5 (Otomat) Bir otomat

$$G=(Q,\Sigma,f,\Gamma,q_0,Q_m) \qquad \textbf{(2.4)}$$

altılısı ile tanımlanır [36]. Burada;

- Q durum kümesi,
- Σ sonlu olay kümesi,
- $f:Q\times\Sigma\to Q$ durum geçiş fonksiyonudur. $f(q,\sigma)=q'$, q durumundan q' durumuna σ olayı ile bir geçişin olduğunu tanımlar. Genel halde f fonksiyonu tanım kümesi üzerinde kısmi bir fonksiyondur. Yani $Q\times\Sigma$ kümesinin tamamında tanımlı olmayabilir,
- $\Gamma:Q\to 2^Q$ aktif olay fonksiyonu adını alır. $\Gamma(q)$, otomat q durumundayken f(q,σ)' nın tanımlı olduğu bütün σ olaylarını içerir ve G otomatının q durumundaki *aktif olay kümesi* olarak adlandırılır. (Bir Q kümesi için 2^Q, Q kümesinin tüm alt kümelerini gösterir)
- q_0 ilk durum,
- $Q_m\subseteq Q$ ise işaretli durumlar kümesidir. ■

Otomatın yukarıda verilen tanımı için *durum makinesi* ismi de verilmektedir. Durum sayısı sonlu ise *sonlu otomat*, ya da *sonlu durum makinesi* isimleri de kullanılmaktadır.

Otomatın tanımı Γ aktif olay fonksiyonu olmadan da tam olarak verilebilir. Çünkü aktif olay fonksiyonunun taşıdığı bilgi, f durum geçiş fonksiyonu tarafından

belirlenebilmektedir. Otomat tanımına ilave edilmesinin nedeni, ileride görüleceği gibi otomat davranışı ile ilgili yapılacak analizlerde kolaylık sağlamasıdır.

Yukarıdaki tanımda f fonksiyonu için belirtilen tanım kümesinde kısmi olma özelliği otomatın davranışı açısından önem taşır. Aşağıda otomat ve dil ilişkisi kurulduğunda görüleceği üzere, bu özellik otomatla ilişkilendirilen L dili ile Σ olay kümesi arasında $L \subseteq \Sigma^*$ ilişkisinin sağlanmasına karşılık gelir.

Hangi durumların işaretli olacağı ise tamamen otomatın modellediği sistem davranışı ile ilgilidir. İşaretli durumlar, modellenen sistemde belirli bir işlemin tamamlanması, davranışta amaçlanan durumlardan birine gelinmesi gibi anlamlar taşır.

Bu tanım ve açıklamalardan sonra otomatın nasıl çalıştığı şu şekilde tariflenebilir: Otomat başlangıçta q_0 durumundadır. $\sigma \in \Gamma(q_0) \subseteq \Sigma$ olayı ile $f(q_0,\sigma) \in Q$ geçiş fonksiyonu ile belirlenen duruma geçilir. Otomatın durum geçişleri tanımlı durum geçişleri ile bu şekilde devam eder.

f durum geçiş fonksiyonunun tanım kümesi $Q \times \Sigma$' dan $Q \times \Sigma^*$' ya aşağıdaki gibi genişletilebilir:

$$f(q,\varepsilon) = q$$
$$f(q,s\sigma) = f(f(q,s),\sigma) \quad s \in \Sigma^* \text{ ve } \sigma \in \Sigma$$

Bu özellik durum geçiş fonksiyonlarında olayların dışında kelimelerin de kullanılmasını mümkün kılmaktadır.

2.4.1 Otomatın temsil ettiği diller

Bir otomatın durum diyagramı gösterimi ile o otomat için tanımlı olan iki dil belirlenebilir. Otomat için mümkün olan tüm geçişlerle elde edilen olay dizilimleri *otomat tarafından üretilen dil*e, bunlardan işaretli durumlarla sonlananlar ise *otomat tarafından işaretlenen dil*e karşılık gelir. Bu dillere ilişkin tanımlar formal olarak aşağıdaki gibi verilebilir. Bu tanımlarda $f: Q \times \Sigma^* \to Q$ şeklinde genişletilmiş durum geçiş fonksiyonu kullanılacaktır.

Tanım 2.6 (Bir otomatın ürettiği dil) $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ tarafından üretilen dil

$$L(G) = \{s \in \Sigma^* : f(q_0,s) \text{ tanımlıdır}\} \qquad \textbf{(2.5)}$$

şeklinde tanımlanır [36]. ■

Tanım 2.7 (Bir otomatın işaretlediği dil) $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ tarafından işaretlenen dil

$$L_m(G) = \{s \in L(G) : f(q_0, s) \in Q_m\} \quad \textbf{(2.6)}$$

olarak tanımlanır [36]. ■

Bir otomatın ürettiği dil içindeki bir s kelimesinin tüm ön ekleri yine üretilen dil kümesinin içinde yer alır. Bu nedenle tanım gereği üretilen dil önek kapalı bir dildir: $L(G) = \overline{L(G)}$. Şekil 2.2' de verilen G otomatı için üretilen ve işaretlenen diller sırasıyla aşağıdaki gibi olacaktır:

$$L(G) = \{\varepsilon, \sigma_1, \sigma_1\sigma_4, \sigma_1\sigma_4\sigma_4, \ldots, \sigma_1\sigma_2, \sigma_1\sigma_4\sigma_2, \sigma_1\sigma_4\sigma_2\sigma_1, \sigma_1\sigma_4\sigma_2\sigma_1\sigma_3, \sigma_1\sigma_4\sigma_2\sigma_1\sigma_1 \ldots\}$$

$$L_m(G) = \{\sigma_1\sigma_2, \sigma_1\sigma_4\sigma_2, \sigma_1\sigma_4\sigma_4\sigma_2, \sigma_1\sigma_4\sigma_2\sigma_1\sigma_1\sigma_2, \sigma_1\sigma_4\sigma_2\sigma_1\sigma_1\sigma_4\sigma_2, \ldots\}$$

Daha önce de belirtildiği gibi, otomat teorisindeki tanımından farklı olarak ayrık olay sistemleri (AOS) teorisinde f durum geçiş fonksiyonu tanım kümesinde kısmi fonksiyondur. f fonksiyonunun bu özelliği otomatın ürettiği dilin Σ^* dil kümesinin bir alt kümesi olmasına neden olur. Bu ise otomatın belirli bir davranışı modellemesi açısından önem taşır. Aksi durumda otomatın ürettiği dil Σ^*' a eşit olacaktır [36,37].

2.4.2 Kilitlenme

Otomatın, üretilen ve işaretlenen dillerin tanımı gereği

$$\overline{L_m(G)} \subseteq L(G) \quad \textbf{(2.7)}$$

ilişkisi her zaman sağlanır. Otomatın bir q durumu için $\Gamma(q) = \phi$ ve $q \notin Q_m$ özellikleri sağlansın. Bu koşullarda otomat q durumuna geldiğinde başka bir duruma geçemeyecektir, çünkü başka bir olayın meydana gelmesi mümkün değildir. Bu durum işaretli bir durum olmadığından, sistem belirli bir görevi yerine getirememiş ve *kilitlenme* söz konusu olmuştur. Otomatta bu özelliklerde durum ya da durumlar olması halinde, bu durumlarla sonuçlanan kelimeler doğal olarak $\overline{L_m(G)}$ dilinde bulunmayacaklardır. Bu ise üretilen dilde, işaretlenen dilin önekinde bulunmayan kelimelerin bulunması anlamına gelir, yani $\overline{L_m(G)} \subset L(G)$ ilişkisi sağlanır.

Bir otomatta bazı durumlar için birbirleri arasında geçiş mümkünken, başka durumlara geçiş olmayabilir. Otomat bu durumlardan birine girdiğinde bunların

dışındaki bir duruma geçemeden çalışmaya devam edecektir. Bu durumlardan hiçbiri işaretli değilse yine *kilitlenme* söz konusudur, çünkü yine tanımlı bir görev tamamlanamamaktadır. İki kilitlenme arasındaki farkı belirtmek üzere ilkine *ölü kilitlenme*, ikincisine *canlı kilitlenme* adı verilir. Her iki tür kilitlenmede de $\overline{L_m(G)} \subset L(G)$ ilişkisi sağlanır. Aşağıda kilitlenmenin matematiksel tanımı verilmiştir.

Tanım 2.8 (Kilitlenme) Bir G otomatı için

$$\overline{L_m(G)} \subset L(G) \qquad \textbf{(2.8)}$$

özelliği sağlanıyorsa *kilitlenmeli,*

$$\overline{L_m(G)} = L(G) \qquad \textbf{(2.9)}$$

özelliği sağlanıyorsa *kilitlenmesiz* otomat denir. ■

2.4.3 Deterministik olmayan otomat

Buraya kadar ele alınan otomat tanımında, belirli bir durumda bir olayın meydana gelmesi ile tek bir duruma geçiş mümkündür. Ayrıca durumlar arası geçiş ε' dan farklı olaylarla ilişkilendirilmiştir. Bu anlamda buraya kadar ele alınan otomat davranışı deterministiktir, yani belirli bir s kelimesine karşılık otomatın durum yörüngesi tam olarak belirlidir. Bu otomat tanımı *deterministik otomat* olarak adlandırılır. Ancak sistem davranışıyla ilgili eksik bilgi nedeniyle otomat gösteriminde yukarıda bahsedilen iki duruma ters düşen modeller elde edilebilir. Bu tür bir otomata ise *deterministik olmayan otomat* adı verilir. Deterministik olmayan otomatta olay kümesi ε ile genişletilmiş, durum geçiş fonksiyonun tanım ve değer kümesi ise sırasıyla $Q \times \Sigma \cup \varepsilon$ ve 2^Q şeklinde değiştirilmiş olur. Bu nedenle deterministik olmayan otomat için aşağıdaki gibi yeni bir tanım verilmelidir:

Tanım 2.9 (Deterministik olmayan otomat) Deterministik olmayan otomat,

$$G = (Q, \Sigma \cup \{\varepsilon\}, f_{nd}, \Gamma, q_0, Q_m) \qquad \textbf{(2.10)}$$

altılısı ile ifade edilmektedir [36]. Burada $f_{nd} : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ deterministik olmayan otomata ilişkin durum geçiş fonksiyonunu ifade eder. Olay kümesi ε'nu da içerecek şekilde genişletilmiştir. Ayrıca deterministik olmayan otomatta ilk durum bir durumlar kümesi olabilir, yani $q_0 \subseteq Q$' dır. ■

Bu çalışmada aksi belirtilmedikçe deterministik otomatlarla ilgilenilecektir.

2.5 Otomatlar Üzerine İşlemler

Ayrık olay sistemlerinin modelleme, kontrol ve tasarım uygulamalarında otomatlar kullanılırken otomatın belirli kurallara göre ve belirli özelliklerini ortaya çıkaracak şekilde değiştirilmesi, birden fazla otomatın bir arada belirledikleri davranışı temsil eden başka bir otomatın elde edilmesi gibi işlemlere gereksinim duyulur. Aşağıda otomatlar üzerine yapılabilecek bu işlemler tanımlanmıştır. Otomat üzerine işlemler tekli ve çoklu işlemler olarak iki başlık altında incelenecektir.

2.5.1 Tekli işlemler

Tekli işlemlerde işleme tek bir otomat girer. Aşağıda bir G otomatı için Erişilebilir Kısım, Kilitlenmesiz Kısım ve Budama işlemleri tanımlanmıştır.

2.5.1.1 Erişilebilir kısım

Bir G otomatında bazı durumlar q_0 durumundan herhangi bir kelime ile erişilemiyor olabilir. Otomattaki bu durumların ve bunlara ilişkin tüm geçişlerin silinmesinin G' nin ürettiği ve işaretlediği dillere bir etkisi olmayacaktır. Bir G otomatının erişilebilir kısmını alma işlemi Ac(G) ile gösterilir ve matematiksel olarak aşağıdaki gibi tanımlanır:

$$\mathrm{Ac}(G) = (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m}) \qquad \textbf{(2.11)}$$

burada,

$$Q_{ac} = \{q \in Q : \exists s \in \Sigma^* \ \ (f(q_0, s) = q)\}$$
$$Q_{ac,m} = Q_m \cap Q_{ac}$$
$$f_{ac} : Q_{ac} \times \Sigma \to Q_{ac}$$

olarak verilir.

2.5.1.2 Kilitlenmesiz kısım

Bir G otomatının durum diyagramındaki bütün durumlardan işaretli durumlara erişilebilen kelime ya da kelimeler bulunuyorsa, G kilitlenmesiz bir otomattır. İşaretli durumlara erişilemeyen durumların varlığı ise otomatın kilitlenmeli olmasına neden

olur. Bir G otomatındaki bu durumların silinerek kilitlenmesiz kısmını alma işlemi CoAc(G) ile gösterilir ve aşağıdaki gibi tanımlanır:

$$\mathrm{CoAc(G)} = (\mathrm{Q_{coac}}, \Sigma,\ \mathrm{f_{coac}}, \mathrm{q_{0,coac}}, \mathrm{Q_m}) \qquad \textbf{(2.12)}$$

burada,

$$\mathrm{Q_{coac}} = \{q \in Q : \exists s \in \Sigma^*\ \ (f(q,s) \in \mathrm{Q_m})\}$$

$$\mathrm{q_{0,coac}} = \begin{cases} q_0 & q_0 \in \mathrm{Q_{coac}}\ \text{ise} \\ \text{tanimsiz} & \text{diger} \end{cases}$$

$$\mathrm{f_{coac}} : \mathrm{Q_{coac}} \times \Sigma \to \mathrm{Q_{coac}}$$

olarak verilir.

2.5.1.3 Budama

Hem erişilebilir hem de kilitlenmesiz bir otomat *budanmış* otomat olarak adlandırılır. Budama işlemiş ise:

$$\mathrm{Trim(G)} = \mathrm{CoAc}[\mathrm{Ac(G)}] = \mathrm{Ac}[\mathrm{CoAc(G)}] \qquad \textbf{(2.13)}$$

olarak tanımlanır.

2.5.2 Çoklu işlemler

Birden fazla otomatın birbirleri ile etkileşerek bir arada oluşturduğu iki farklı ortak davranış türü tanımlanabilir. Bu ortak davranışlar otomatların *çarpım* ve *paralel birleşim* işlemleri ile tanımlanır ve sırasıyla "$\times$" ve "$||$" sembolleri ile gösterilirler. Paralel birleşim *senkron birleşim*, çarpım ise *tam senkron birleşim* isimleri ile de anılabilmektedir.

Bu işlemler,

$\mathrm{G_1} = (\mathrm{Q_1}, \Sigma_1, \mathrm{f_1}, \Gamma_1, \mathrm{q_{01}}, \mathrm{Q_{m1}})$ ve $\mathrm{G_2} = (\mathrm{Q_2}, \Sigma_2, \mathrm{f_2}, \Gamma_2, \mathrm{q_{02}}, \mathrm{Q_{m2}})$ otomatlarının erişilebilir olduğu varsayımı ile aşağıdaki gibi tanımlanır:

2.5.2.1 Çarpma

$\mathrm{G_1}$ ve $\mathrm{G_2}$ otomatlarının çarpımı

$$\mathrm{G_1} \times \mathrm{G_2} = \mathrm{Ac}(\mathrm{Q_1} \times \mathrm{Q_2}, \Sigma_1 \cap \Sigma_2, \mathrm{f}, \Gamma_{1\times 2}, (\mathrm{q_{01}}, \mathrm{q_{02}}), \mathrm{Q_{m1}} \times \mathrm{Q_{m2}}) \qquad \textbf{(2.14)}$$

ile verilir. Burada yeni durum geçiş fonksiyonu,

$$f((q_1,q_2),\sigma)=\begin{cases}(f_1(q_1,\sigma),f_2(q_2,\sigma)) & \sigma\in\Gamma_1(q_1)\cap\Gamma_2(q_2)\\ \text{tanimsiz} & \text{diger}\end{cases}$$

şeklide tanımlanır, dolayısıyla $\Gamma_{1\times2}(q_1,q_2)=\Gamma_1(q_1)\cap\Gamma_2(q_2)$ 'dir.

Tanımdan görüldüğü gibi çarpım sonucu elde edilen otomatta bir olayın olması için o olayın her iki otomatın da olay kümesinde bulunması ve her iki otomatta da eşzamanlı olarak meydana gelmesi gerekir. Dil açısından $G_1\times G_2$ ile elde edilen davranış, her iki otomatın tam olarak senkronize olduğu davranışı verir. $G_1\times G_2$ otomatının durumları ilk bileşeni G_1'in, ikinci bileşeni G_2'nin olmak üzere durum çifti olarak ifade edilir. Çarpım ile elde edilen otomatın ürettiği ve işaretlediği diller ile çarpıma giren otomatların dilleri arasında kolaylıkla gösterilebilecek aşağıdaki ilişki sağlanır:

$$\begin{aligned}L(G_1\times G_2)&=L(G_1)\cap L(G_2)\\ L_m(G_1\times G_2)&=L_m(G_1)\cap L_m(G_2)\end{aligned}\qquad\textbf{(2.15)}$$

Buradan, $\Sigma_1\cap\Sigma_2=\phi$ ise $L(G_1\times G_2)=\varepsilon$ ve (q_{01},q_{02}) durumunun işaretli olup olmamasına bağlı olarak $L_m(G_1\times G_2)=\varepsilon$ ya da $L_m(G_1\times G_2)=\phi$ olacağı sonucu çıkarılabilir.

2.5.2.2 Paralel birleşim

G_1 ve G_2 otomatlarının paralel birleşimi aşağıdaki gibi tanımlanır.

$$G_1\,||\,G_2=\mathrm{Ac}(Q_1\times Q_2,\Sigma_1\cup\Sigma_2,f,\Gamma_{1||2},(q_{01},q_{02}),Q_{m1}\times Q_{m2})\qquad\textbf{(2.16)}$$

Burada,

$$f((q_1,q_2),\sigma)=\begin{cases}(f_1(q_1,\sigma),f_2(q_2,\sigma)) & \sigma\in\Gamma_1(q_1)\cap\Gamma_2(q_2)\\ (f_1(q_1,\sigma),q_2) & \sigma\in\Gamma_1(q_1)\setminus\Sigma_2\\ (q_1,f_2(q_2,\sigma)) & \sigma\in\Gamma_2(q_2)\setminus\Sigma_1\\ \text{tanimsiz} & \text{diger}\end{cases}$$

olarak verilir. Buradan $\Gamma_{1||2}(q_1,q_2)=[\Gamma_1(q_1)\cap\Gamma_2(q_2)]\cup[\Gamma_1(q_1)\setminus\Sigma_2]\cup[\Gamma_2(q_2)\setminus\Sigma_1]$ olduğu gösterilebilir.

Görüldüğü gibi her iki otomatın olay kümesinde bulunan bir olayın olması için çarpma işleminde olduğu gibi bu olayın her iki otomatta da eşzamanlı olarak meydana gelmesi gerekir. Bunun dışında bir otomatta bulunup, diğerinde

bulunmayan olaylar, yani $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$ kümesinin elemanları için eşzamanlı olma koşulu aranmamaktadır.

Paralel birleşim ile elde edilen dilin paralel birleşime giren otomatların dilleri cinsinde ifadesinde *dillerin izdüşümü* kavramı kullanılır. Kelimeler için

$$P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*, \qquad i = 1,2$$

izdüşümleri aşağıdaki gibi tanımlanmaktadır:

$$\begin{aligned} &P_i(\varepsilon) = \varepsilon \\ &P_i(\sigma) = \begin{cases} \sigma & \sigma \in \Sigma_i \text{ ise} \\ \varepsilon & \sigma \notin \Sigma_i \text{ ise} \end{cases} \\ &P_i(s\sigma) = P_i(s)P_i(\sigma), \quad s \in (\Sigma_1 \cup \Sigma_2)^*, \sigma \in (\Sigma_1 \cup \Sigma_2) \end{aligned} \tag{2.17}$$

Yukarıdaki tanıma göre, $\Sigma_1 \cup \Sigma_2$ kümesinin elemanları ile oluşturulmuş bir s kelimesinin Σ_i (i=1,2) kümesine izdüşümü, s'ten izdüşüm yapılan küme dışındaki olayların silinmesi ile elde edilir. Bu izdüşüm, *doğal izdüşüm* olarak adlandırılır.

Bu tanımdan hareketle

$P_i^{-1} : \Sigma_i^* \to 2^{(\Sigma_1 \cup \Sigma_2)^*}$ *ters izdüşüm*ü ise

$$P_i^{-1}(t) : \{s \in (\Sigma_1 \cup \Sigma_2)^* : P_i(s) = t\} \tag{2.18}$$

olarak tanımlanır [36].

İzdüşüm ve ters izdüşüm tanımları dildeki tüm kelimelere uygulanarak dillere de genişletilebilir. Bir $L \subseteq (\Sigma_1 \cup \Sigma_2)^*$ dili için izdüşüm,

$$P_i(L) = \{t \in \Sigma_i^* : \exists s \in L\,(P_i(s) = t)\} \tag{2.19}$$

bir $L \subseteq \Sigma_i^*$ dili için ters izdüşüm ise

$$P_i^{-1}(L_i) = \{s \in (\Sigma_1 \cup \Sigma_1)^* : \exists t \in L_i\,(P_i(s) = t)\} \tag{2.20}$$

olarak tanımlanır.

İzdüşüm ve ters izdüşüm için $P_i[P_i^{-1}(L)] = L$ sağlanır, ancak genel durumda $L \subseteq P_i^{-1}[P_i(L)]$'dir.

Aşağıda bir L dili için izdüşüm ve ters izdüşüm örneği verilmektedir:

$\Sigma_1 = \{\sigma_1, \sigma_2\}$ ve $\Sigma_2 = \{\sigma_2, \sigma_3\}$ olsun. $L = \{\sigma_1, \sigma_1\sigma_3, \sigma_2\sigma_3\sigma_3, \sigma_3\sigma_2\sigma_2\sigma_1\sigma_3\}$ için

$P_1(L) = \{\sigma_1, \sigma_2, \sigma_2\sigma_2\sigma_1\}$
$P_2(L) = \{\varepsilon, \sigma_3, \sigma_2\sigma_3\sigma_3, \sigma_3\sigma_2\sigma_2\sigma_3\}$
$P_1^{-1}(\{\varepsilon\}) = \{\sigma_3\}^*$
$P_1^{-1}(\{ab\}) = \{\sigma_3\}^*\{a\}\{\sigma_3\}^*\{b\}\{\sigma_3\}^*$

olacaktır.

İzdüşüm ve ters izdüşüm tanımları ile G_1 ve G_2 otomatlarının paralel birleşimi için üretilen ve işaretlenen dillerin aşağıdaki gibi olduğu gösterilebilir:

$$L(G_1 \,||\, G_2) = P_1^{-1}[L(G_1)] \cap P_2^{-1}[L(G_2)]$$
$$L_m(G_1 \,||\, G_2) = P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2)] \qquad \textbf{(2.21)}$$

Otomatlar için paralel birleşim işlemi (2.21) ilişkisinden yararlanarak diller için de aşağıdaki gibi tanımlanabilir.

$$L_1 \,||\, L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2) \qquad \textbf{(2.22)}$$

Bir otomattaki her duruma, o otomatta bulunmayan diğer olayların özçevrim olarak eklenmesi ile ters izdüşüm işlemi gerçeklenebilir. Bu şekilde elde edilen yeni otomatın ürettiği ve işaretlediği diller özçevrim eklenmeden önce üretilen ve işaretlenen dillerin ters izdüşümü olacaktır. Buradan hareketle (2.21) ifadelerinin karşılığı G_1 ve G_2 otomatlarının her durumuna sırasıyla $\Sigma_1 \setminus \Sigma_2$ ve $\Sigma_2 \setminus \Sigma_1$ olaylarını öz çevrim olarak ekleyip, elde edilen otomatlarla çarpma işleminin uygulanması olarak değerlendirilebilir.

Daha önce belirtildiği gibi bu çalışmada durum diyagramı gösterimine dayalı yöntemler, özel olarak otomat gösterim biçimi temel alınacaktır. Ancak burada, ayrık olay sistemlerinin modellenmesi, tasarımı ve kontrolünde yaygın olarak kullanılan bir diğer gösterim biçimi olan Petri Ağları da genel özellikleri ile tanıtılacaktır.

2.6 Petri Ağları

Petri ağları bir ilk durum ile beraber tanımlanan, ağırlık değeri atanmış yönlü okların iki farklı tür bileşeni ilişkilendirdiği, iki parçalı (bipartite), yönlü bir graf türüdür. Bu graftaki yönlü oklar dışındaki bileşenler *yer* (place, p) ve *geçiş* (transition, t) olarak adlandırılırlar. Yönlü oklar yerlerden geçişlere, ya da geçişlerden yerlere tanımlı

olabilir. Grafiksel gösterimde yerler dairelerle, geçişler ise çubuk ya da kutularla ifade edilir. Oklar, ağırlıkları ile beraber gösterilirler. Ağırlık değeri pozitif tam sayılarla ifade edilir. Ağırlık değeri görünmüyorsa, değeri birdir. Ağırlığa ilişkin birden büyük bir *k* değeri *k* adet paralel yönlü oka karşılık gelir. Belirli bir durumda bir *p* yerine *n* tamsayısı atanmışsa, *p* yeri *n* adet jetonla *işaretlenmiştir* denir. İşaretleme, grafiksel olarak ilgili yere jeton sayısı kadar içi dolu daireler konmasıyla gösterilir. Bir Petri ağının işaretlemesi (durumu) yerlere atanmış negatif olmayan tam sayılarla belirlenir ve *m* boyutlu bir **x** vektörü ile ifade edilir. Burada *m* yerlerin sayısıdır. **x** vektörünün *p*. bileşeni *x(p)* ile gösterilir ve bu değer *p*. yerdeki jeton sayısına karşılık gelir. Aşağıda Petri ağlarının formal tanımı verilmektedir:

Tanım 2.10 (Petri Ağları) Bir Petri ağı:

$$(P,T,A,w,\mathbf{x}_0) \qquad \textbf{(2.23)}$$

beşlisi ile tanımlanır. Burada;

- P sonlu *yer*ler kümesi,
- T sonlu *geçiş*ler kümesi,
- $A \subseteq (P \times T) \cup (T \times P)$ yerlerden geçişlere ve geçişlerden yerlere tanımlı oklar kümesi,
- $w : A \rightarrow \{1,2,3,...\}$ oklara ilişkilendirilmiş ağırlık fonksiyonu,
- $\mathbf{x}_0$ ilk işaretlemedir. ■

Bir ilk işaretleme içermeyen *N*=(P,T,A,w) dörtlüsü, Petri ağı grafı ya da yapısı olarak adlandırılır. Bir $\mathbf{x}_0$ ilk işaretlemesi tanımlanmış *N* Petri ağı $(N,\mathbf{x}_0)$ ile gösterilir.

Yerler $P = (p_1, p_2, ..., p_n)$, geçişler $T = (t_1, t_2, ..., t_m)$ ile gösterilecektir. Oklar yerden geçişe, ya da geçişten yere tanımlı olmalarına göre sırasıyla (p_i, t_j) ve (t_j, p_i) şeklinde gösterilir. Bir oka ilişkin ağırlık, pozitif tam sayılarla ifade edilir ve p_i yerinden t_j geçişine yönlenmiş okun ağırlığı $w(p_i, t_j)$ ile ifade edilir. Bir Petri ağındaki bir geçiş için bazı yerler giriş, bazı yerler ise çıkış olarak nitelendirilebilir. Bir t_j geçişi için giriş olan yerler $I(t_j)$, çıkış olan yerler ise $O(t_j)$ ile gösterilir. Yukarıda da belirtildiği gibi belirli bir andaki $\mathbf{x} = [x(p_1), x(p_2), ..., x(p_n)]$ işaretlemesi, Petri ağı grafında yerlere konan sırasıyla $x(p_1)$, $x(p_2)$ ve $x(p_n)$ adet jetonla ifade edilir. Esasen, bu işaretleme modelleme açısından otomatlardaki durum

kavramına karşılık gelir. Bu nedenle Petri ağına ilişkin bir **x** işaretlemesi Petri ağının durumu olarak anılacaktır. Sonuç olarak bir Petri ağının X durum uzayı, elemanları negatif olmayan tam sayılar olan tüm n boyutlu vektörlerden oluşur, dolayısıyla X = $\mathbf{N}^n$ şeklinde gösterilebilir.

Ayrık olay sistemlerin modellenmesi açısından, Petri ağlarında olaylar geçişlerle ilişkilendirilmiştir. Ancak, bir olayın meydana gelmesi koşullara bağlıdır. Bir geçişe ilişkin olayın meydana gelme koşulu, bu geçiş için giriş olan yerler tarafından belirlenir. Geçiş için çıkış olan yerler ise olayın meydana gelmesinden etkilenirler ve bu yerler tarafından belirlenen koşullar güncellenmiş olur. Koşullar Petri ağının işaretlemesi ile ifade edilir. Bir $t \in T$ geçişine ilişkin koşul, bu geçişe giriş olan her bir yerlerdeki jeton sayısının, en az bu yerleri t'ye bağlayan okların ağırlıkları kadar olmasıdır. Bunun formal ifadesi aşağıdaki gibi verilir.

Tanım 2.11 (İzinli geçiş) Bir Petri ağındaki $t_j \in T$ geçişi

$$x(p_i) \geq w(p_i, t_j), \quad \forall p_i \in I(t_j) \qquad \textbf{(2.24)}$$

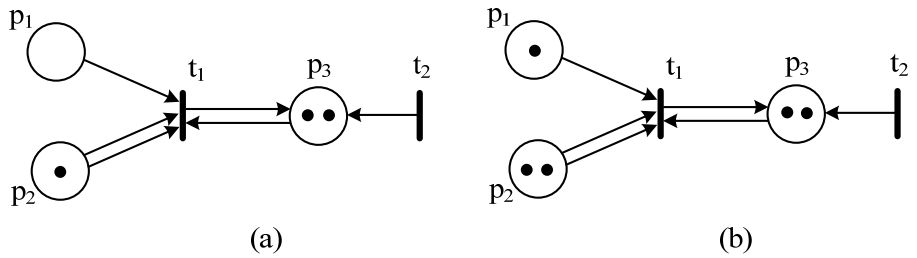
koşulu sağlanıyorsa izinlidir denir. ■

Şekil 2.3'te örnek bir Petri ağı iki farklı işaretlemesi ile verilmektedir. Bu Petri ağının bileşenleri şu şekildedir:

$P = \{p_1, p_2, p_3\}$, $T = \{t_1, t_2\}$, A={ (p_1, t_1), (p_2, t_1), (t_1, p_3), (p_3, t_1), (t_2, p_3) },

$w(p_1, t_1) = 1$, $w(p_2, t_1) = 2$, $w(t_1, p_3) = 1$, $w(p_3, t_1) = 1$, $w(t_2, p_3) = 1$

Şekil 2.3a'daki işaretleme $\mathbf{x}_1 = [0 \quad 1 \quad 2]$ ile, Şekil 2.3b'deki işaretleme ise $\mathbf{x}_2 = [1 \quad 2 \quad 2]$ verilir. $\mathbf{x}_1$ işaretlemesinde t_1 geçişi izinli değildir. Çünkü t_1 geçişi için giriş olan yerlerde $x(p_1) = 0 < w(p_1, t_1)$ ve $x(p_2) = 1 < w(p_2, t_1)$' dir. $\mathbf{x}_2$ işaretlemesi içinse j=1 için (2.24) koşulu sağlandığından t_1 izinlidir.



Şekil 2.3 : Örnek bir Petri ağı ve buna ilişkin iki farklı işaretleme (durum)

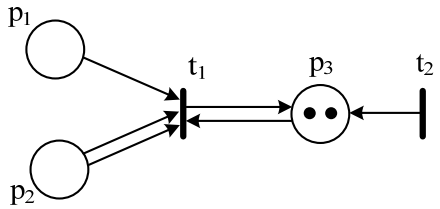
Buraya kadar Petri ağlarındaki geçişlerin, dolayısıyla geçişlere karşılık gelen olayların izinli olmalarından bahsedilmiştir. Geçişlerin izinli olması gerçekleşeceği anlamına gelmez. Geçişlerin izinli olması ilgili olayların meydana gelmesi için ön koşuldur. Bu ise otomat gösteriminde, belirli bir durumun aktif olay kümesine karşılık gelmektedir. Otomat gösteriminde aktif olay kümesindeki bütün olaylar bir sonraki adımda meydana gelecek olay için birer adaydır. Bir Petri ağında ise belirli bir işaretlemeye (duruma) karşılık birden fazla geçiş izinli olabilir. Sonuçta bu izinli geçişlere karşılık gelen olaylardan biri meydana gelir ve Petri ağı için yeni bir işaretlemeye (duruma) geçilmiş olur. Bu yeni işaretlemeye ilişkin kural ise Petri ağının tanımladığı dinamiğe karşılık gelir ve formal olarak Tanım 2.12 ile verilir.

Tanım 2.12 (Petri ağlarının dinamiği) $(P,T,A,w,\mathbf{x}_0)$ ile verilen bir Petri ağının durum geçiş fonksiyonu $f: \mathbf{N}^n \times T \rightarrow \mathbf{N}^n$ 'nin tanımlı olabilmesi için gerek ve yeter koşul (2.24) ifadesi ile verilir. Eğer $f(\mathbf{x},t_j)$ tanımlıysa $\mathbf{x}' = f(\mathbf{x},t_j)$ olmak üzere yeni işaretleme (durum)

$$x'(p_i) = x(p_i) - w(p_i,t_j) + w(t_j,p_i), \quad i = 1,...,n \qquad \textbf{(2.25)}$$

ile belirlenir.

(2.25) ifadesine göre p_i, t_j geçişi için bir giriş yeri ise, geçişin gerçekleşmesi ile p_i 'den t_j 'ye tanımlı okun ağırlığı kadar jeton kaybeder. Tersi olarak bir çıkış yeri ise t_j 'den p_i 'ye tanımlı okun ağırlığı kadar jeton kazanır. p_i bir t_j geçişi için hem giriş hem de çıkış yeri olabilir. Bu durumda p_i, t_j geçişinin gerçekleşmesi ile $w(p_i,t_j)$ kadar jeton kaybeder, $w(t_j,p_i)$ kadar jeton kazanır. Şekil 2.3b'deki Petri ağında t_1 geçişinin gerçekleşmesiyle oluşan yeni işaretleme (durum) Şekil 2.4'te görülmektedir.



Şekil 2.4 : t_1 geçişinin gerçekleşmesinden sonra Şekil 2.3b'deki Petri ağına ilişkin yeni durum.

Bu noktada Şekil 2.4'te görülen t_2 geçişi için giriş yerinin bulunmadığına dikkat edilmelidir. Bu tür geçişlere ilişkin olaylar koşulsuz olarak geçekleşebilmektedir.

Görüldüğü üzere Petri ağlarında yerlerdeki jetonlar, Petri ağının modellediği ayrık olay sisteminin durumlarına karşılık gelmektedir. Belirli bir geçiş sayısı sonunda Petri ağının durumu $x = [0,...,0]$ olabilir. Ya da aksine geçişler yerlerdeki jeton sayısının giderek büyümesine neden olabilir. Bir Petri ağındaki yer sayısı sonlu olmasına rağmen, yerlerdeki jeton sayısı için bir üst sınırın bulunmaması sonlu bir yapı ile sonsuz durumlu bir davranışın modellenebileceği anlamını taşır. Bu ise Peti ağları ile otomat gösterimi arasındaki temel farklılıklardan biridir. Tanım gereği otomat sonlu durumlu olduğundan ancak sonlu duruma karşılık gelen davranışlar modellenebilmektedir. Bu nedenle bir otomata karşılık Petri ağı gösteriminin oluşturulabilmesine rağmen, tersi her zaman mümkün değildir. Sonuç olarak Petri ağları ile daha geniş bir dil kümesi ifade edilebilmektedir.

Belirli geçişlerden sonra bir Petri ağında hiçbir geçiş izinli olmayabilir. Bu Petri ağının yeni bir işaretlemeye ya da duruma geçememesi anlamını taşır. Bunun Otomat gösterimindeki karşılığı ise kilitlenmedir.

Petri ağlarına ve Petri ağı türlerine ilişkin literatürde yaygın çalışma bulunmaktadır. Petri ağlarına ve türlerine ilişkin detaylı bilgiler için [37,38], Petri ağlarının ayrık olay sistemlerinin kontrolündeki uygulamaları için ise [16,25,26,28-30] kaynakları görülebilir.

Buraya kadar zamansız dil modeli ve buna karşılık gelen zamansız modellerle ilgili temel kavramlar tanıtılmıştır. İlerleyen bölümlerde verilen bir otomata karşılık gelen davranışın teknolojik araçlarla gerçeklenmesi detaylı olarak irdelenecektir. Gerçekleme ile ilgili önemli konulardan biri belirli zamanlama davranışlarının modellenebilmesidir. Her ne kadar bu çalışmada gerçekleme aşaması için kullanılmış olmasa da, aşağıda zamanlı otomat modeli ve PLC ile gerçeklemeye uygun bir otomat türü olarak literatürde PLC-otomat olarak adlandırılan otomat modeli irdelenecek, ilerleyen bölümlerde zamanlama davranışlarına ilişkin nasıl bir çözüm getirildiği bu otomat modelleri ile karşılaştırmalı olarak verilecektir.

2.7 Zamanlı Otomat

Ayrık olay sistemlerinin modellenmesi için kullanılan standart otomat gösterimi olayların oluş zamanları için bir bilgi içermez. Eğer modelde olayların olma zamanı ile de ilgileniliyorsa $\{\sigma_1,\sigma_2,...\}$ şeklinde olay dizileri ya da $\{q_0,q_1,...\}$ şeklinde durum dizileriyle beraber $\{(q_0,t_0),(q_1,t_1),...\}$ şeklinde durum ve durum geçişlerine ilişkin zamanları da içermesi gerekir.

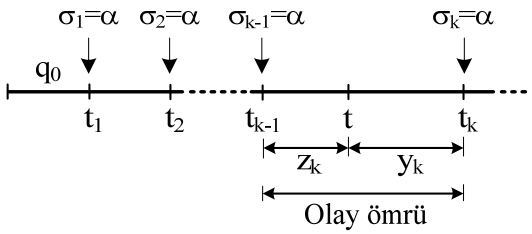
İlk aşamada, zamanlı otomat için (2.4) ifadesinde, Σ olay kümesinin sonsuz olmasına izin verecek ve kilitlenme ile ilgilenilmediğinden işaretli durum tanımı çıkartılarak (Q,Σ,f,Γ,q_0) otomat modeli ele alınacaktır. Zamanlı otomat tanımını tamamlamak için olayların zamana bağlı davranışını tanımlayan "Saat Yapısı" kavramı tanıtılmalıdır. Bu kavram aşağıda basit örnekler üzerinden anlatılmıştır [36]. Zamanlı otomat teorisiyle ilgili kapsamlı bilgi için [40] görülebilir.

2.7.1 Saat yapısı

Zamanlama mekanizmasını tanımlamak için gerekli kavramları açıklayabilmek üzere aşağıda tek olaylı bir AOS örneği ele alınacaktır. Bir genellemeye varmak için daha karmaşık örneklerle devam edilecektir [36].

$\Sigma=\{\alpha\}$ ve her $q\in Q$ için $\Gamma(q)=\{\alpha\}$ olsun. Bu durumda bu AOS için olay-zaman diyagramı Şekil 2.5'teki gibi verilebilir. Bu olay-zaman diyagramına ilişkin olay dizilimi $\sigma_k=\alpha$, k=1,2,… olmak üzere $\{\sigma_1,\sigma_2,...\}$ ile gösterilebilir. Olayların oluş anları ise t_k, k=1,2,… ile gösterilmektedir. İki olay arasında geçen süreye *olay ömrü* adı verilmektedir. Buna göre olayın k. ömrü aşağıdaki gibi tanımlanır.

$$v_k=t_k-t_{k-1} \quad k=1,2,\ldots \qquad \textbf{(2.26)}$$



Şekil 2.5 : $\Sigma=\{\alpha\}$ olay kümeli bir AOS için olay-zaman diyagramı [36]

Bu sistemin zamana bağlı davranışı şu şekilde tanımlanabilir. t_{k-1} anında k. olay *aktiftir*, ya da *izinlidir* denilir ve bu olay için v_k olay ömrü verilmiş olur. Tam bu

anda bu olaya ilişkin bir saat kurulur ve geri saymaya başlar. $t_k = t_{k-1} + v_k$ anında olay ömrü tamamlanır ve olay meydana gelir. Bu da bir durum değişikliğine neden olacak ve (k+1). olay aktif hale gelecektir. Sistem, bu şekilde devamlı olarak kendisini tekrar edecektir.

Bu noktada olayın oluşuyla aktif hale gelmesi arasındaki farkı vurgulamak gerekir. Olayın aktif olmasından kasıt otomatın mevcut durumunun aktif olay kümesi içerisinde olmasıdır. Bu olaya ilişkin saatin sıfıra ulaşması ile olay meydana gelir ve bir durum geçişine neden olur. Şekil 2.5'teki α olayı her zaman aktiftir, ancak sadece t_1, t_2, … anlarında meydana gelmektedir. Bu örnek için α olayının her meydana gelişinde yeniden aktive olmaktadır, çünkü her durumda α olayı aktif olay kümesi içinde yer alır. Şekil 2.5'ten de görüleceği üzere v_k olay ömrü

$$v_k = z_k + y_k \qquad \textbf{(2.27)}$$

ile verilebilir. Açıktır ki olay ömrü dizisi $\{v_1, v_2, \ldots\}$ verildiğinde bu AOS'nin olay-zaman diyagramı tam olarak tanımlanmış olur. Bu diziye aynı zamanda α olayının *saat dizisi* adı da verilmektedir.

Sürekli olarak aktif iki olayı olan bir AOS örneği için olay kümesi $\Sigma = \{\sigma, \beta\}$ olan bir sistem ele alınacaktır. Kolaylık açısından her $q \in Q$ için $\Gamma(q) = \{\sigma, \beta\}$ olduğu, yani her iki olayın her an aktif olduğu varsayılacaktır. Her bir olaya ilişkin saat dizileri $\mathbf{v_\alpha} = \{v_{\alpha,1}, v_{\alpha,2}, \ldots\}$ ve $\mathbf{v_\beta} = \{v_{\beta,1}, v_{\beta,2}, \ldots\}$ olarak verilmiş olsun. Olay-zaman diyagramı Şekil 2.6'da görülen böyle bir AOS'nin $v_{\alpha,1}$ ve $v_{\beta,1}$ değerlerinin karşılaştırılması ile bir t_0 anında bir sonraki olayın hangisinin gerçekleşeceğinin belirlenebilir. Örneğin,

$$v_{\alpha,1} < v_{\beta,1} \qquad \textbf{(2.28)}$$

koşulu sağlanıyorsa $t_1 = t_0 + v_{\alpha,1}$ anında olacak olay α' dır. t_1 anından sonra meydana gelecek ilk olayın belirlenmesi içinse saat değerlerinin yeniden karşılaştırılması gerekecektir. Ancak β olayı hala aktiftir ve yeni saat değeri

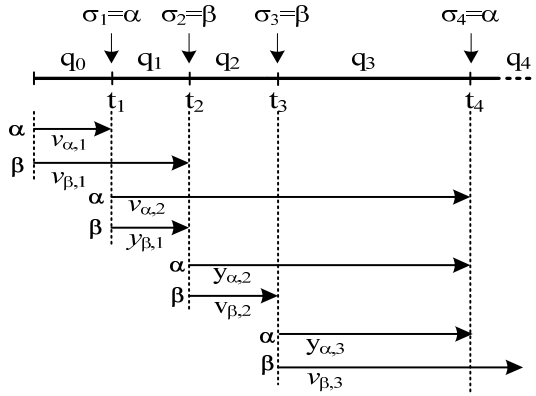
$$y_{\beta,1} = v_{\beta,1} - v_{\alpha,1} \qquad \textbf{(2.29)}$$

ile hesaplanır. Bu durumda bir sonraki olayın belirlenmesi için $v_{\alpha,2}$ ile $y_{\beta,1}$' in karşılaştırılması yapılmalıdır. Eğer,

$$y_{\beta,1} < v_{\alpha,2} \qquad \textbf{(2.30)}$$

ise $t_2 = t_1 + y_{\beta,1}$ anında meydana gelecek olay β'dır.

Bir sonraki olayın belirlenmesi sonuç olarak saat değerlerinin karşılaştırılması ve en küçük olanın seçilmesine dayanmaktadır. Eğer bir olay henüz olmuşsa, saat değeri saat dizisindeki bir sonraki değerle güncellenir.



Şekil 2.6 : $\Sigma = \{\alpha,\beta\}$, olayları sürekli aktif olan AOS için olay-zaman diyagramı [36]

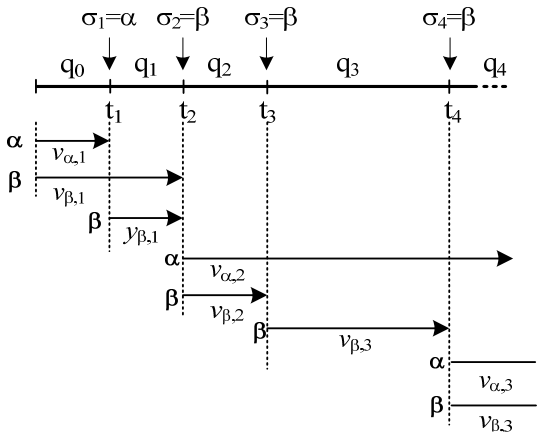
Son olarak, Şekil 2.7'deki gibi olayların tümünün her durumda aktif olmadığı AOS yapısı incelenecektir. Bir önceki örnekten farklı olarak bazı $q \in Q$ için $\Gamma(q) = \{\alpha,\beta\}$ ve diğer $q \in Q$ için $\Gamma(q) = \{\beta\}$ olduğu varsayılacaktır. q_0 ilk durumu için $\Gamma(q_0) = \{\alpha,\beta\}$ olsun. Bu durumda bir önceki örnekteki gibi ilk olacak olay α' dır. Yeni q_1 durumu için $\Gamma(q_1) = \{\beta\}$ olsun. Bu durumda tek bir olay aktif olduğu için saat değerlerinin karşılaştırılmasına gerek yoktur. t_2 anında β olayı olur ve q_2 durumuna geçilir. Bu durumda her iki olayın da aktif olay kümesi içinde olduğunu varsayılsın. Her iki olay da yeni saat değerlerini alırlar $v_{\beta,2}<v_{\alpha,2}$ olduğundan bir sonraki olay β'dır. Geçilen yeni q_3 durumunda $\Gamma(q_3) = \{\beta\}$ olsun. Bu durumda $t = t_4$ anında β olayı olur ve q_4 durumuna geçilir. $\Gamma(q_4) = \{\alpha,\beta\}$ olduğu varsayılırsa her iki olay da aktif olur. Bu durumda $t = t_3$ anında α olayı deaktive edildiğinden $v_{\alpha,2}$ geçerliliğini kaybeder ve α için tamamen yeni bir $v_{\alpha,3}$ saat değeri atanır. Sonuç olarak sistem davranışı olayların aktif olmalarına ve güncellenen saat değerlerini karşılaştırılmasına bağlı olarak belirlenmiş olur.

Bu örneklerden yola çıkılarak olayların oluş zamanlarına ilişkin kuralı oluşturmak üzere aşağıda zamanlı AOS'ler için *saat yapısı* (clock structure) tanımı verilmiştir.

Tanım 2.10 (Saat Yapısı): Σ olay kümesine ilişkin *Saat Yapısı* (*Zamanlama Yapısı*)

$$V=\{\mathbf{v_i} : i \in \Sigma\},\ \mathbf{v_i}=\{v_{i,1}, v_{i,2},\ldots\},\ i \in \Sigma,\quad v_{i,k} \in \mathbf{R}^+,\ k=1,2,\ldots \qquad \textbf{(2.31)}$$

ile verilen olay ömrü dizi kümesidir [36].



Şekil 2.7 : α olayının her durumda aktif olmadığı olay-zaman diyagramı [36]

Saat yapısı ele alınan AOS için bir giriş olarak düşünülecektir. Saat yapısı bilgisiyle ele alınan AOS'ye ilişkin otomatın üreteceği olay dizisi elde edilir. Bu nedenle verilen bir AOS için saat yapısı V'nin de tam olarak belirlenmiş olduğu varsayılacaktır.

Bir diğer kullanışlı kavram ise olay skorudur.

Tanım 2.11 (Olay Skoru): Bir $i \in \Sigma$ olayına ilişkin $N_{i,k}$ ile gösterilen olay skoru, olay-zaman diyagramında k. durum geçişinden sonra $[t_0, t_k]$ aralığında *i* nin kaç kez aktive olduğunu gösteren sayıdır [36].

Örneğin Şekil 2.6'da t_0 anında her iki olay da aktive edildiğinden olay skorları $N_{\alpha,0} = N_{\beta,0} = 1$ olarak belirlenmiştir. t_1 anında $N_{\alpha,1} = 2$ ve $N_{\beta,1} = 1$ olur. t_2 anında ise olay skorları $N_{\alpha,2} = N_{\beta,2} = 2$ olarak belirlenir. Sonuç olarak her olay oluşunda olaylardan yeniden aktive olanların bulunup ilgili skorların 1 artırılması gerekir. Bir *i* olayının skoru, bu olayın bir sonraki aktive oluşunda atanacak olay ömrünün belirlenmesi için $\mathbf{v_i}$ saat dizisine ilişkin bir işaretçi olarak kullanılır.

Aşağıda şu ana kadar bahsedilen olay-zaman ilişkisi için formal bir tanım verilmeye çalışılacaktır. Gösterilim kolaylığı açısından şu notasyon kullanılacaktır:

q mevcut durum

σ q durumuna geçişe neden olan en son olay

t σ olayına karşılık gelen en son olayın oluş anı

Ayrıca her $i \in \Sigma$ olayına ilişkin iki değişken daha tanımlanacaktır:

N_i *i* olayının mevcut skoru, $N_i \in \{0,1,\ldots\}$

y_i *i* olayının mevcut saat değeri, $y_i \in \mathbf{R}^+$

Bir sonraki durum, olay, olay oluş anı, olay skoru ve saat değeri için (') notasyonu kullanılacaktır. Buna göre q , q_k ya, q' ise q_{k+1} e karşılık gelecektir.

Zamanlı otomatın formal tanımı aşağıdaki gibi verilir:

Tanım 2.12 (Zamanlı Otomat): $\mathbf{V}=\{\mathbf{v_i} : i \in E\}$ saat yapısı ve (X, E, f, Γ, x_0) bir otomat olmak üzere *Zamanlı Otomat*

$$(Q, \Sigma, f, \Gamma, q_0, \mathbf{V}) \qquad \textbf{(2.32)}$$

altılısı ile tanımlanır [36]. Bu otomat

$$\sigma' = \arg\min_{i \in \Gamma(q)} \{y_i\}$$

kuralı ile üretilen ve $\{\sigma_1, \sigma_2, ..\}$ olay dizisi ile sürülen $q' = f(q, \sigma')$ durum dizlerini üretir. Olaylar, değeri her adımda güncellenen

$$y'_i = \begin{cases} y_i - y^* & i \neq \sigma' \text{ ve } i \in \Gamma(q) \\ v_{i,N_i+1} & i = \sigma' \text{ veya } i \notin \Gamma(q) \end{cases} \qquad i \in \Gamma(q')$$

saat değerlerine bağlı olarak üretilir. Burada y^* olaylar arası süresi

$$y^* = \min_{i \in \Gamma(q)} \{y_i\}$$

$i \in \Sigma$ olmak üzere N_i olay skorları,

$$N'_i = \begin{cases} N_{i+1} & i = \sigma' \text{ veya } i \notin \Gamma(q') \\ N_i & \text{diger} \end{cases} \qquad i \in \Gamma(q')$$

ile tanımlanır. İlk değerler ise $i \in \Gamma(q_0)$ için $y_i = v_{i,1}$ ve $N_i = 1$; $i \notin \Gamma(q_0)$ için y_i tanımsız ve $N_i = 0$ olarak belirlenir. ■

2.8 PLC-otomat

PLC-otomat ayrık olay sistem varsayımlarından farklı olarak durum geçişlerini olaylar yerine *giriş*lere ilişkilendirmektedir. PLC-otomatta durumlara gecikme süreleri atayarak otomata bir tür gecikme davranışı kazandırmak mümkündür. PLC-otomat sıfırdan farklı gecikme tanımlanan bir duruma geçtiğinde, bu duruma ilişkin gecikme süresi doluncaya kadar durum geçişlerine belirli koşullara bağlı olarak izin

verilmez. Gecikme süresi dolduktan sonra girişlere bağlı durum geçişi gerçeklenebilir. Gecikme süresi tanımlanan bir durumda, süre dolmasa bile, bazı girişlere karşılık gelen durum geçişlerinin engellenmesi istenmeyebilir. Gecikme süresinden etkilenmesi istenmeyen bu girişlerin ayıt edilebilmesi için her bir duruma gecikme süresince otomatın "duyarsız" kalacağı girişler kümesi atanır. Bu küme dışından bir girişin gelmesi ile o duruma ilişkin gecikme süresi dolmasa bile durum geçişi gerçekleşecektir.

Formal olarak PLC-otomat aşağıdaki şekilde tanımlanır:

Tanım 2.13 (PLC-otomat): Bir PLC-otomat

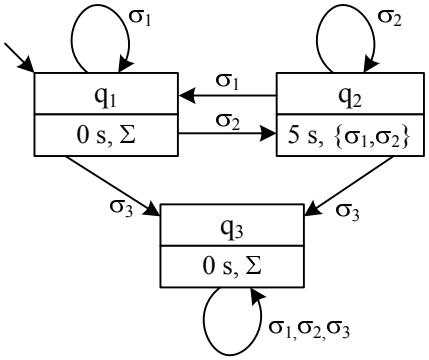
$$A = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega) \qquad \textbf{(2.33)}$$

dokuzlusu ile verilir [41]. Burada,

- Q sonlu, boş olmayan durum kümesi,
- Σ sonlu, boş olmayan girişler kümesi,
- $\delta : Q \times \Sigma \to Q$ durum geçiş fonksiyonu,
- q_0 ilk durum,
- $\varepsilon > 0$ bir çevrim için üst sınır,
- $S_t : Q \to \mathbf{R}_{\geq 0}$ her q durumuna $S_e(q)$ ile tanımlanan girişlerin ne kadar süre duyarsız kalınacağını atayan fonksiyon,
- $S_e : Q \to P(\Sigma)$ her q durumuna $S_t(q)$ saniye boyunca durum değişikliğine neden olmayacak şekilde geciktirilmiş girişler kümesini atayan fonksiyon,
- Ω boş olmayan sonlu çıkışlar kümesi,
- $\omega : Q \to \Omega$ çıkış fonksiyonudur. ■

Şekil 2.8'de örnek bir PLC-otomat verilmektedir. Bu otomata ilişkin durum kümesi $Q = \{q_1, q_2, q_3\}$, giriş kümesi $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ şeklindedir. Görüldüğü gibi q_1 durumunda gecikme süresi sıfırdır, yani bu durumdaki girişler herhangi bir gecikme olmadan durum geçiş fonksiyonu ile tanımlanan geçişlerle sonuçlanacaktır. Bu durumdayken σ_1 girişinin gelmesi ile PLC-otomat q_2 durumuna geçer. Bu durumda σ_1 ve σ_2 girişleri için 5 s'lik bir gecikme tanımlanmıştır. PLC-otomat, q_2 durumuna geçtikten sonraki 5 s'lik süre boyunca σ_1 ve σ_2 girişlerine duyarsızdır. 5 s sonunda σ_2 girişi gelmeye devam ettiği süre boyunca q_2 durumunda kalacak, σ_1 işareti ise q_1 durumuna geçişe neden olacaktır. Otomat q_2 durumundayken 5 s'lik sürenin

dolmasına bakılmaksızın σ_3 girişinin gelmesi q_3'e geçişe neden olur. Çünkü $\sigma_3 \notin S_e(q_2)$' dir.



Şekil 2.8 : Örnek bir PLC-otomata ilişkin durum diyagramı [41]

3. AYRIK OLAY SİSTEMLERİNİN KONTROLÜ

Ayrık olay sistemleri (AOS) olarak sınıflandırılabilecek sistemler, bir önceki bölümde belirtildiği gibi, farklı alanlarda, genellikle karmaşıklığı yüksek olan dinamik sistemler olarak ortaya çıkmaktadır. Bu tür sistemlerin kontrol uygulamalarının geliştirilmesinde *sezgisel* ya da *formal* olarak iki farklı yaklaşımdan bahsedilebilir. Sezgisel yaklaşım sistemin kontrolü, yani sisteme istenilen davranışın kazandırılması için herhangi bir model kullanılmaksızın, tecrübeye ve deneme-yanılma süreçlerine dayalı, kontrolörün tasarlanması ve uygulanmasının genellikle eşzamanlı olarak yapıldığı yöntemlere dayanır. Formal yaklaşımda ise istenilen davranışa karşılık gelen kural tanımları altında tasarım matematiksel olarak yapılır ve çeşitli araçlar kullanılarak gerçek dünyada uygulanır. Uygulamada sezgisel yöntemlerin çok yaygın olarak kullanılmasına rağmen, ayrık olay sistemlerinin karmaşıklığının artması modelleme ve formal tasarım süreçlerini içeren yaklaşımların kullanılmasını zorunlu kılmaktadır.

Tarsımda kullanılabilecek formal yöntemler de, çeşitli sınıflandırmalarla ele alınabilir. Çoğu durumda ele alınan AOS'i ifade eden bir model olmaksızın, istenilen kontrol davranışına karşılık gelen tasarımın ifadesi durum diyagramları, Petri ağları gibi gösterim biçimleri, ya da mantıksal fonksiyonlar ile elde edilir. Diğer bir formal tasarım yöntem sınıfı olarak ise kontrol teorisinde olduğu gibi kontrol edilecek sistem modeli ve istenilen kontrol davranışından yola çıkılarak, sisteme istenilen davranışı kazandıracak tasarım yöntemleri uygulanır ve tasarımın ifadesi yine bir gösterim biçimiyle elde edilir. Bu anlamda Ramadge ve Wonham tarafından önerilen *üstdenetim kuramı (supervisory control theory)*, ayrık olay sistemlerinin kapalı çevrimli kontrolü ve *üstdenetleyici (supervisor)* tasarımı için kuramsal bir yapı oluşturmaktadır.

Sonuçta, ele alınan formal tasarım yönteminin sistem modeline dayalı olup olmamasından bağımsız olarak, tasarım bir gösterim biçimi ile ifade edilir. Bir sonraki aşama ise, bir gösterim biçimi ile ifade edilen soyut tasarımla aynı davranışı gerçek dünyada sergileyecek bir uygulamanın geliştirilmesinden oluşur.

Bu bölümde ayrık olay sistemlerinin kontrolünde modele dayalı bir formal yöntem olarak Üstdenetim Kuramı ve bu kurama ilişkin farklı kontrol yapıları tanıtılacaktır. Modele dayalı olmayan formal yöntemler de özetlenecek ve gerçekleme için uygulamada sıklıkla kullanılan Programlanabilir Lojik Kontrolörlere (PLC) ilişkin temeller verilecektir. PLC programlarında kullanılan formal yöntemler ve bunların karşılaştırılması ile ilgili kapsamlı bir çalışma için [42] görülebilir. Bu çalışmanın gerçekleme konusundaki katkı ve uygulamalarına ilerleyen bölümlerde yer verilecektir.

3.1 Üstdenetim Kuramı

Üstdenetim Kuramında kontrol teorisindeki yaklaşıma benzer bir şekilde kontrol edilmek istenen sistem ve kontrolör iki ayrı yapı olarak ele alınır ve kontrolörün görevi (burada üstdenetleyici) geribesleme ile sistem davranışını sistemden beklenen davranışa getirmektir.

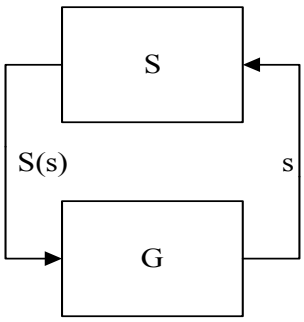
Üstdenetim Kuramında sistem davranışı sistemin ürettiği dil, sisteme istenilen dilin kazandırılması ise sistem davranışını ürettiği dilin bir alt kümesine sınırlandırılması anlamını taşır. Uygulamadan buna farklı örnekler vermek mümkündür. Örneğin bir G otomatı ile modellenen bir ayrık olay sisteminin kontrol edilmeyen davranışında belirli bir kurala uygun olmadığı için belirli olay dizilimleri istenmiyor olabilir. Ya da G otomatının belirli durumları kilitlenmeye neden olduğu için bu durumlara erişilmesi istenmeyebilir. Başka bir örnek ise üretim bantlarındaki bir ara ürünün başka bir ara ürün ile bir çarpışmaya neden olacak şekilde aynı yere ürerim bileşenleri tarafından aynı yere yerleştirilmesini engellemek olabilir.

Üstdenetim Kuramında sistem davranışının istenen davranış kümesine sınırlandırılması *S* ile gösterilecek olan bir *üstdenetleyici* tarafından, geribesleme ile sağlanır. Çoğu durumda, istenilen davranışa karşılık gelen dil için tek bir alt dil yerine bir alt dil aralığı ile ilgilenilmektedir. Kontrol ile elde edilecek olan L dilinin bulunduğu bu dil aralığı, L_r istenilen en küçük dil kümesi, L_a izin verilen en büyük dil kümesi ve L(G) ele alınan sistemin kontrolsüz davranışı olmak üzere

$$L_r \subseteq L \subseteq L_a \subseteq L(G) \qquad \textbf{(3.1)}$$

şeklinde tanımlanır [36].

$G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ otomatına ilişkin olay kümesi G otomatının ürettiği dil $L(G) = \overline{L(G)}$ ve işaretlediği dil $L_m(G)$ olsun. G otomatının S gözetmen kontrolörü ile Şekil 3.1' deki yapıya uygun olarak şu şekilde çalışır: G'nin ürettiği her olay, S tarafından gözlemlenir. S üstdenetleyicisi bir takım olayların olmasını engelleme yeteneğine sahiptir. G sisteminin ürettiği her olaydan sonra olmasına izin verilen olaylar kümesi ve olması yasaklanan olaylar kümesi güncellenir ve bir sonraki adımda G sisteminin üretebileceği ancak S tarafından izin verilmeyen olayların olması engellenir. Böylece sistemin kontrolü için dinamik bir geribesleme yapısı elde edilmiş olur [1,36]. Bazı olayların olmasının üstdenetleyici tarafından engellenmesi hiçbir şekilde mümkün değildir. Bu olaylara *kontrol edilemeyen* olaylar denir ve olmalarını engellemek mümkün olmadığından her zaman izinlidirler. Bazı olayların kontrol edilemeyen olaylar olarak modellenmesinin çeşitli nedenleri olabilir. Örneğin bir algılayıcıdan alınan sinyale karşılık düşen bir olay kontrol edilemeyen bir olaydır, çünkü algılayıcıdan alınan sinyal üstdenetleyicinin doğrudan bir etkisi mümkün değildir. Çalışan bir sistemin arızaya geçmesi de kontrol edilemeyen olaylara bir örnek olabilir.



Şekil 3.1 : Üstdenetleyicili kontrol yapısı

G sistemi ve S gözetmeninin bir arada oluşturduğu kapalı çevrimli sistem S/G, G' nin S gözetmeni altında ürettiği dil $L(S/G)$, işaretlediği dil ise $L_m(S/G)$ gösterilir.

Tanım 3.1 (S/G' nin ürettiği ve işaretlediği diller) L(S/G) dili aşağıdaki gibi iki aşamada tanımlanır:

$$\varepsilon \in L(S/G)$$
$$[s \in L(S/G) \text{ ve } (s\sigma \in L(G) \text{ ve } (\sigma \in S(s))] \Leftrightarrow [s\sigma \in L(S/G)] \quad \textbf{(3.2)}$$

işaretlenen dil ise

$$L_m(S/G) = L(S/G) \cap L_m(G) \quad \textbf{(3.3)}$$

olarak verilir. ■

Tanıma göre $L(S/G) \subseteq L(G)$ dili önek kapalıdır: $L(S/G) = \overline{L(S/G)}$. Ayrıca genel durumda şu ilişki sağlanır:

$$\phi \subseteq L_m(S/G) \subseteq \overline{L_m(S/G)} \subseteq L(S/G) \subseteq L(G) \quad \textbf{(3.4)}$$

S/G için kilitlenme ile ilgili tanımlar ise aşağıdaki gibi verilmektedir:

Tanım 3.2 (**Kontrol edilen sistemde kilitlenme)** Bir S/G kapalı çevrimli sistemi için

$$L(S/G) \neq \overline{L_m(S/G)} \quad \textbf{(3.5)}$$

ise kilitlenmeli,

$$L(S/G) = \overline{L_m(S/G)} \quad \textbf{(3.6)}$$

ise kilitlenmesizdir denir. ■

G otomatı ile modellenen bir ayrık olay sistemine ilişkin Σ_{uc} kümesi ile verilen kontrol edilmeyen olayların bulunması durumunda ayrık olay sisteminden beklenen kontrol davranışını sağlayan bir üstdenetleyicinin bulunması her zaman mümkün olmayabilir. Üstdenetleyicinin kontrol edilemeyen olayları engelleyememesinden dolayı, üretilen dil kontrol edilemeyen olaylarla istenmeyen kelimeler içerilecek şekilde genişleyebilir ve hedeflenen sistem davranışından farklı bir dil elde edilebilir. Bu nedenle, bir G ve Σ_{uc} için istenilen kontrol davranışını sağlayacak bir S üstdenetleyicisinin varlığına ilişkin koşul tanımlanmalıdır. Bu koşul aşağıdaki *kontroledilebilirlik teoremi* ile verilir.

Teorem 3.1 (Kontroledilebilirlik Teoremi)

Bir $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$ ayrık olay sistemi, $\Sigma_{uc} \subset \Sigma$ kontrol edilemeyen olaylar kümesi, $K \subseteq L(G)$ $(K \neq \phi)$ kontrol altında sistemden beklenen davranışlara karşılık gelen dil kümesi olsun. Buna göre $L(S/G) = \overline{K}$'yı sağlayacak bir S üstdenetleyicisinin varlığı için gerek ve yeter koşul

$$\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K} \quad \textbf{(3.7)}$$

ile verilir [1,36]. ■

K dili üzerindeki bu koşul kontrol edilebilirlik koşulu olarak adlandırılmaktadır. Bu koşulun sağlanması durumunda sistemden beklenen davranışın $S(s) = [\Sigma_{uc} \cap \Gamma(f(q_0, s))] \cup \{\sigma \in \Sigma_c : s\sigma \in \overline{K}\}$ şeklinde tanımlanan kontrolör ile kazandırılabileceği gösterilebilir.

Tanım 3.3 (Bir dilin kontroledilebilirliği) K ve $M = \overline{M}$, Σ olay kümesi üzerinde tanımlı diller ve Σ_{uc}, Σ ' nın bir alt kümesi olsun. Bu durumda K dili için

$$\overline{K}\Sigma_{uc} \cap M \subseteq \overline{K} \qquad \textbf{(3.8)}$$

koşulu sağlanıyorsa M ve Σ_{uc} 'ya göre kontroledilebilirdir denir [1,36]. ■

S/G sisteminin kilitlenmesiz olmasını sağlayacak bir S üstdenetleyicisinin varlık koşulu ise aşağıdaki teorem ile verilir.

Teorem 3.2 (Kilitlenmesiz Kontroledilebilirlik Teoremi)

Bir $G = (Q, \Sigma, f, \Gamma, q_0, Q_m)$, $\Sigma_{uc} \subset \Sigma$ kontrol edilemeyen olaylar kümesi, $K \subseteq L(G)$ $(K \neq \phi)$ kontrol altında sistemden beklenen davranışlara karşılık gelen dil kümesi olsun. Buna göre Buna göre $L_m(S/G) = K$ ve $L(S/G) = \overline{K}$ 'yı sağlayacak bir S üstdenetleyicisinin varlığı için gerek ve yeter koşul

$$\begin{aligned} &\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K} \text{ ve} \\ &K = \overline{K} \cap L(G) \end{aligned} \qquad \textbf{(3.9)}$$

ilişkilerinin aynı anda sağlanmasıdır [10,36]. ■

3.1.1 Kontroledilebilir alt ve üst diller

G otomatı ile modellenen bir ayrık olay sistemi ve Σ_{uc} kontrol edilemeyen olaylar kümesi için üstdenetimle kazandırılmak istenen K dili, kontroledilebilir olmayabilir. Bu durumda üstdenetleyici tasarımı için K dili yerine (3.1) ilişkisi ile verilen aralıktaki bir çözüme karşılık gelecek şekilde kontroledilebilir alt dillerle ile ilgilenilir. Bu alt diller, bir K dili için *kontroledilebilir alt dillerin supremumu* ve *önek kapalı kontroledilebilir üst dillerin infimumu* olarak adlandırılır ve sırasıyla $K^{\uparrow C}$ ve $K^{\downarrow C}$ ile gösterilir. $K^{\uparrow C}$ dili, $K^{\uparrow C} \subseteq K$ sağlanacak şekilde L(G) ve Σ_{uc} ' ya

göre K'nın kontroledilebilir olan alt kümelerinin en büyüğüdür. Bir üstdenetim problemi için K dili kontroledilebilir değilse, kontroledilebilir olan alt kümelerinin en büyüğünün aranması *üstdenetimle kontrolün temel problemi* (basic supervisory control problem-BSCP) olarak adlandırılır [36]. Doğal olarak $K^{\uparrow C}$ bu problemin çözümüdür. $K^{\downarrow C}$ dili ise verilen bir K dilinden büyük olan önek kapalı ve kontrol edilebilir en küçük dildir. Kontroledilebilir alt dil olarak (3.1) ilişkisinde verilen L_r dilinden büyük ve "mümkün olan en küçük dil"in aranması problemi ise *üstdenetimle kontrolün ikiz problemi* (Dual version of BSCP) olarak adlandırılır [36]. Bu problemin çözümü de doğal olarak $L_r^{\downarrow C}$ olacaktır.

$K^{\uparrow C}$ ve $K^{\downarrow C}$ dilerinin bulunmasına yönelik algoritmalar ve kontroledilebilirlikle konusundaki detaylar için [1,2,36] kaynakları görülebilir.

3.1.2 Kontrol davranışının tanımlanması

Daha önce belirtildiği gibi bir ayrık olay sisteminin kontrolü, davranışını ürettiği ya da işaretlediği dilin bir alt kümesine sınırlandırılmasıyla sağlanmaktadır. İstenilen davranış aşağıdaki gibi üretilen ve/veya işaretlenen diller açısından aşağıdaki gibi alt dil aralığı ile verilir:

$$L_r \subseteq L(S/G) \subseteq L_a \subset L(G) \text{ ve/veya } L_{rm} \subseteq L_m(S/G) \subseteq L_{am} \subset L(G)$$

L_a dili burada *kabul edilebilir alt dil* olarak isimlendirilecek ve önek kapalı olduğu varsayılacaktır. L_{am} ise *işaretlenen dilin kabul edilebilir alt dili*dir. L_r *istenilen dil*, L_{rm} *istenilen işaretli dil* olarak adlandırılacaktır.

Üstdenetleyicilerin tasarımı için sisteme ilişkin G otomat modelinin yanında L_a, L_{am}, L_r, L_{rm} dillerini temsil eden otomatlara da ihtiyaç duyulur. Uygulamada kontrolle G otomatına kazandırılması istenen davranış kuralları genellikle dilsel ifadelerle verildiğinden bu dilleri üreten ya da işaretleyen otomatların oluşturulması gerekir. Bu dilsel ifadelere şu örneklerin verilmesi mümkündür: "G' nin belirli bir yasaklı durumlar listesine girmesinden kaçın, ilk gelen-ilk alır prensibini uygula, a ve b olaylarının sırasını değiştir, o ana kadar oluşturulan kelimede c olayını ancak b' den önce geliyorsa işle…". Örnekler çoğaltılabilir. Verilen bu dil ifadelerine karşılık gelen otomatların doğrudan elde edilmesi kolay değildir ve çoğunlukla tecrübe gerektirir.

Kontrol davranışını temsil edecek bir otomat modelinin bulunması genellikle iki aşamadan oluşur. Öncelikle sadece istenilen davranışı temsil eden basit bir otomat oluşturulur. Bu otomat H_{ist} olarak gösterilsin. İkinci aşamada H_{ist}, çarpım ya da paralel birleşim işlemi uygulanarak G ile birleştirilir. Elde edilen otomat H_a ile gösterilirse $L(H_a) = L_a$ olacaktır. H_{ist} ve G' nin birleştirilmesinde uygulanacak olan işlem şu şekilde belirlenir: G' de bulunan ama H_{ist}' te bulunmayan olaylar H_{ist}' ile temsil edilen kontrol davranışı ile ilgili değilse paralel birleşim uygulanır. Diğer taraftan bu olaylar L_a' da bulunmaması gerektiğinden H_{ist}'te yoksa çarpma işlemi kullanılır. Genel durumda i=1,2,..,n olmak üzere $H_{ist,i}$ ile ifade edilen birden fazla davranış kuralının bulunabilir. Aşağıda temel birkaç kontrol davranışı örneğinin otomat karşılıklarının nasıl elde edildiği anlatılmaktadır [36]. Bu konudaki daha kapsamlı uygulama örnekleri için [19,23,32] kaynakları görülebilir.

3.1.2.1 Yasaklı durumlar

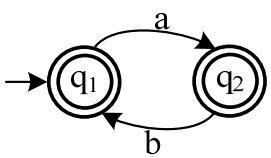
Bir koşul G'nin bazı durumlarını yasaklı olarak tanımlıyorsa, bu durumlar ve o durumlarla ilgili geçişler silinip ve erişilebilir kısmı alınırsa H_a otomatı elde edilebilir. Eğer koşul kilitlenme olmamasını da gerektiriyorsa ilgili durumlar silindikten sonra erişilebilir kısmını almak yerine budama işlemi uygulanmalıdır.

3.1.2.2 Durum bölme

Bir koşul için, G'nin belirli bir durumuna nasıl erişildiği önem taşıyorsa, o durumun gerektiği kadar bölünmesi ve yeni oluşturulan durumların aktif olay kümelerinin o durumdan sonraki yasaklı olmayan davranışa göre belirlenmesi gerekir.

3.1.2.3 Olay dönüşümü

İlk olay a ikinci olay b olmak üzere a ve b gibi iki olayın dönüşümlü olması gerekiyorsa, bunu sağlayan iki durumlu bir H_{ist} otomatı aşağıdaki gibi oluşturulabilir. İstenen H_a otomatı ise $H_a = H_{ist} \,||\, G$ ile bulunabilir.



Şekil 3.2 : Olay dönüşümü için kullanılacak H_{ist} otomatı

Burada paralel birleşim işleminin kuralları a' nın ikinci kez olmasını G tarafından b' nin işlenmesine kadar engelleyecektir. Aynı durum b olayı için de geçerlidir. G' nin diğer olayları H_{ist}' te olmadıkları için etkilenmeyecektir. H_{ist}' in iki durumu da işaretlidir, çünkü bu koşul kilitlenmeyi içermemektedir ve H_a' daki işaretli durumlar G tarafından belirlenecektir.

3.1.3 Üstdenetleyicilerin otomatlarla gerçeklenmesi

Buraya kadar istenilen davranışa karşılık gelen otomatlara ve bunların nasıl elde edinildiğine değinilmiştir. Üstdenetleyiciler ise $S: L(G) \to 2^{\Sigma}$ şeklinde G'nin ürettiği dilden Σ'nın altkümelerine tanımlana bir fonksiyon olarak ele alınmıştır. Kontroledilebilirlik teoremine göre ise $L(S/G) = \overline{K}$ 'yi sağlayan S üstdenetleyicisi

$$S(s) = \left[\Sigma_{uc} \cap \Gamma(f(q_0, s))\right] \cup \left\{\sigma \in \Sigma_c : s\sigma \in \overline{K}\right\}$$

ile tanımlanmaktadır. Bu şekilde üstdenetleyici fonksiyonu her $s \in L(S/G)$ için izinli olan S(s) olay kümeleri ile ifade edilmektedir. Ancak bu ifade pratik açıdan uygun değildir ve S fonksiyonu için bir gösterime ihtiyaç duyulur. Bu gösterim kontrol edilen sistem davranışının modellenmesinde olduğu gibi otomatlarla sağlanacaktır. S üstdenetleyicisinin otomat gösterimi "S'nin bir gerçeklemesi" olarak isimlendirilir. Bu noktada üstdenetleyicilerin gerçeklenmesi ile ilgili iki farklı kavramı vurgulamakta fayda vardır. Burada üstdenetleyicilerin otomatlarla ifadesi irdelenmektedir. Ancak çalışmanın genelinde "gerçekleme" tasarımı yapılmış ve otomat gösterimi elde edilmiş bir üstdenetleyicinin fiziksel araçlar kullanılarak gerçek dünyada uygulanması anlamında kullanılacaktır.

S üstdenetleyicisinin bir otomat gerçeklemesini oluşturmak için $\overline{K}$ dilini işaretleyen bir otomat oluşturmak yeterlidir. Aşağıdaki budanmış R otomatı bu şekilde elde edilmiş olsun:

$$R = (Y, \Sigma, g, \Gamma_R, y_0, Y_m) \qquad \textbf{(3.10)}$$

R için $L_m(R) = L(R) = \overline{K}$ sağlanır. Eğer R, G'ye çarpma işlemi ile "bağlanırsa" elde edilen $R \times G$ otomatı tam olarak aranan S/G kapalı çevrim davranışını verir:

$$L(R \times G) = L(R) \cap L(G) = \overline{K} \cap L(G) = \overline{K} = L(S/G)$$
$$L_m(R \times G) = L_m(R) \cap L_m(G) = \overline{K} \cap L_m(G) = L(S/G) \cap L_m(G) = L_m(S/G)$$

R ile G' nin olay kümeleri aynı olduğu için $R \times G = R \,||\, G$ olacaktır. Yani üstdenetleyicilerin otomatla gerçeklenmesinde paralel birleşim işlemi de kullanılabilmektedir. Bu şekilde elde edilen R gösterimine S üstdenetleyicisinin *standart gerçeklemesi* adı verilir [36].

Buraya kadar üstdenetim kuramına ilişkin kontrol edilebilir dil ve kilitlenmesiz kontrol edilebilir dil için üstdenetleyicinin varlık koşulları ve istenilen bir kontol davranışına karşılık gelen üst denetleyicilerin tasarımı ve otomatla gerçeklenmesine yönelik temel kavramlar verilmiştir. İstenilen dilin kontrol edilebilir ve kilitlenmesiz olduğunun belirlenmesi "doğrulama" (verification) olarak isimlendirilir. Ayrık olay sistemleri için uygulamadaki en büyük zorluk kontrol edilecek sistem için elde edilen otomat gösterimlerinin durum uzaylarının genellikle çok büyük olmasıdır. Çok büyük ölçekli olmayan bir ayrık olay sistemi için bile durum uzayı kolaylıkla 10^{20} mertebesine ulaşabilir. Bu ise doğrulama algoritmalarının çalışmasını imkânsızlaştırmaya kadar varan problemlere neden olmaktadır. Bu nedenle ayrık olay sistemlerinin kontrolünde hem doğrulama, hem de kontrol etmede karşılaşılan güçlükleri ortadan kaldırmak üzere sisteme veya kontrolöre ilişkin modüler veya hiyerarşik model yapıları kullanılmakta ve bu yeni yapılar üzerinde doğrulama algoritmaları geliştirilmektedir. Bu tür yapılar üstdenetleyicinin gerçeklemesinde de bellek kullanımı açısından önemli kazançlar sağlamaktadır. Aşağıda üstdenetim kuramının farklı sistem yapılarına yönelik temeller verilmektedir.

3.1.4 Modüler kontrol

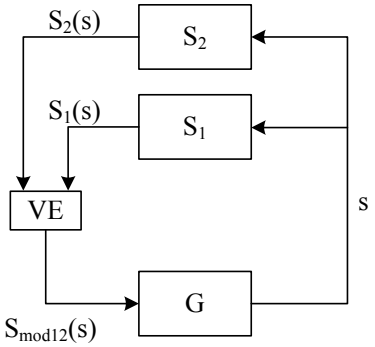
Modüler kontrol, bir S üstdenetleyicisi ile elde edilen kontrol davranışının birden fazla sayıda üstdenetleyicinin bir arada sağladığı kontrol davranışı ile elde edilmesi ilkesine dayanır. Kolaylık olması açısından aşağıda her biri bir G sistemi için tanımlı olan S_1 ve S_2 gibi iki üstdenetleyicinin kullanıldığı bir modüler kontrol uygulaması ele alınacaktır. Modüler üstdenetleyici $S_{mod12} : L(G) \rightarrow 2^{\Sigma}$ şu şekilde tanımlanır:

$$S_{mod12}(s) = S_1(s) \cap S_2(s) \qquad \textbf{(3.11)}$$

Bu tanımdan görüldüğü gibi S_{mod12} ile bir olaya izin verilmesi için o olayın hem S_1 hem de S_2 tarafından izinli olması gerekir. Ya da bir olayın engellenmesi için o olayın S_1 ya da S_2' nin biri tarafından engellenmesi yeterlidir. Modüler kontrol altında kapalı çevrim davranışı

$$L(S_{mod12} / G) = L(S_1 / G) \cap L(S_2 / G)$$
$$L_m(S_{mod12} / G) = L_m(S_1 / G) \cap L_m(S_2 / G) \quad \textbf{(3.12)}$$

şeklindedir. Aşağıda iki üstdenetleyiciden oluşan modüler kontrol mimarisi verilmektedir.



Şekil 3.3 : İki üstdenetleyici ile oluşturulan bir modüler kontrol mimarisi

S_1 ve S_2 için standart gerçeklemeler sırasıyla R_1 ve R_2 olsun. S_{mod12} için standart gerçekleme $R = R_1 \times R_2$ ile oluşturulabilir. Ancak modüler kontrolde amaç R gerçeklemesini oluşturmak değil, R ile aynı davranışı vermek üzere R_1 ve R_2 gerçeklemelerini bir arada kullanmaktır. Buna S_{mod12} üstdenetleyicisinin modüler gerçeklemesi adı verilir [36].

Modüler kontrolde, üstdenetimle kontrolün temel probleminde ele alınan L_a dilinin $L_a = L_{a1} \cap L_{a2}$ gibi iki önek kapalı dilin kesişimi olarak verilmesi durumunda $L_{ai}^{\uparrow C}$, $i = 1,2$ dillerine karşılık gelen S_i üstdenetleyicilerinin tasarımı ile ilgilenilir. Önek kapalı diller için

$$(K_1 \cap K_2)^{\uparrow C} = K_1^{\uparrow C} \cap K_2^{\uparrow C} \quad \textbf{(3.13)}$$

ilişkisi sağlandığından $L_a^{\uparrow C}$ yerine $L_{ai}^{\uparrow C}$, $i = 1,2$ hesabının yapılarak aynı davranışı veren üstdenetleyicilerin tasarlanması mümkündür. Böylece üstdenetleyici tasarımındaki işlemsel karmaşıklık en kötü hal için $O(n_1 n_2 m)$ den $O(\max(n_1, n_2)m))$' e indirgenmiş olur. Ancak kilitlenmesiz modüler kontrol için durum böyle değildir. S_{mod12} ' nin kilitlenmesiz olması için gerek ve yeter koşul

$$\overline{L_m(S_1 / G) \cap L_m(S_2 / G)} = \overline{L_m(S_1 / G)} \cap \overline{L_m(S_2 / G)} \quad \textbf{(3.14)}$$

ya da genel halde

$$\overline{\bigcap_i L_m(S_i / G)} = \bigcap_i \overline{L_m(S_i / G)} \qquad \textbf{(3.15)}$$

şeklinde verilir. Bu koşul görüldüğü gibi $L_{am1}^{\uparrow C}$ ve $L_{am2}^{\uparrow C}$ ile beraber $(L_{am1} \cap L_{am2})^{\uparrow C}$'nin hesaplanmasını gerektirecektir. Bu anlamda kilitlenmesiz modüler kontrol işlemsel karmaşıklık açısından tek bir üstdenetleyicili durumla aynı mertebededir. Ancak yine de bellekten önemli kazanım sağlayabileceğinden uygulamalarda kullanılabilir.

Modüler kontrolle ilgili daha fazla bildi için [4,36] kaynakları görülebilir.

3.1.5 Lokal modüler kontrol

Görüldüğü gibi modüler kontrolde üstdenetleyicinin modüler olması esas alınır. Ancak uygulamada kontrol edilen sistemin modeli de alt sistem modellerinin birleşimi ile elde edilir ve genellikle elde edilen model durum sayısı açısından oldukça büyüktür. Bu nedenle kontrol edilecek sistemin modülerliğinin de kullanıldığı modüler kontrol uygulamaları söz konusudur. Literatürde *Lokal Modüler Üstdenetim* olarak adlandırılan uygulamaya ilişkin *lokal modüler olma koşulu* aşağıdaki gibi tanımlanır [5,6]:

$$\overline{\|_{j=1}^{m} L_m(S_{lok,j} / G_{loc,j})} = \|_{j=1}^{m} \overline{L_m(S_{lok,j} / G_{loc,j})} \qquad \textbf{(3.16)}$$

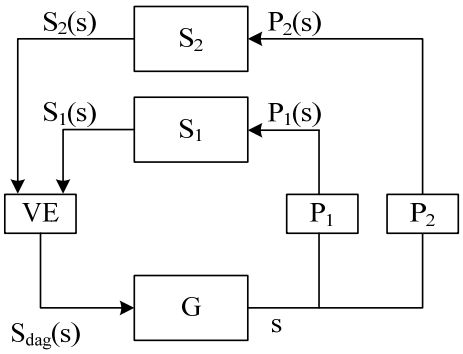
(3.16) ifadesinde paralel birleşimin (2.22) ile verilen dillere uyarlanmış tanımı kullanılmıştır.

3.1.6 Dağıtılmış kontrol

Teknolojik sistemlerin çoğunda yapısal olarak dağıtılmış bir sistemi beraberce kontrol eden işlem noktaları bulunur. Birbirlerine konveyörlerle bağlı çok sayıda işlem noktası bulunan karmaşık bir üretim sistemi örnek olarak düşünülebilir. Haberleşme ve bilgisayar ağları da farklı noktalardaki kontrolörler tarafından dağıtılmış bir yapıda kontrol edilen sistemler olarak ele alınabilir. Bu yapıyı modüler kontrol yapısından ayıran temel özellik dağıtılmış üstdenetleyicilerin sistemdeki tüm olay dizilerini değil de bunların bir izdüşümünü izliyor olması ve her bir üstdenetleyicinin gözlenebilir ve kontrol edilebilir olay kümelerinin farklı olmasıdır.

Şekil 3.4' te dağıtılmış kontrole ilişkin yapı verilmiştir. Bu yapıya göre S_1 ve S_2 üst denetleyicileri sistemin ürettiği olay dizilimlerini değil de, bu olay dizilimlerinin

bulundukları işlem noktalarına karşılık gelen izdüşümlerini izleyebilmektedir. Dağıtılmış kontrol yaklaşımında üstdenetleyicilerin farklı gözlenebilir ve kontrol edilebilir olay kümeleri altında kilitlenmesiz kontroledilebilirlik koşulları tanımlanmaktadır [8-11].



Şekil 3.4 : Dağıtılmış kontrol yapısı

3.1.7 Hiyerarşik kontrol

Ayrık olay sistem modellerinin durum uzayının çok büyük olması, gerçekleme aşaması kadar, kontroledilebilir ve kilitlenmesiz olma özelliklerinin belirlenmesinde, yani doğrulama aşamasında da temel problemi oluşturmaktadır. Bu nedenle modelin bütününün değil de, belirli özellikleri taşıyan bileşenlerine uygulanan doğrulama işlemi ile genel davranışa ilişkin sonuçları verecek yöntemler üzerinde çalışılmaktadır. Bunlardan biri de kontrol edilecek sistem ve üstdenetleyiciyi birbirleri ile *arayüz*ler üzerinden etkileşimde bulunan hiyerarşik seviyelere bölerek doğrulama algoritmalarını bu hiyerarşik seviyelere uygulayan *hiyerarşik kontrol* yapısıdır. Bu yapıda hiyerarşik seviyeler için *seviyeye bağlı kilitlenmesiz* olma *(level-wise nonblocking)* ve *seviyeye bağlı kontroledilebilir* olma *(level-wise controllable)* tanımları verilmekte ve belirli koşullar altında seviyeye bağlı kilitlenmesiz ve kontroledilebilir olmanın genel kilitlenmesiz ve kontroledilebilir olmayla eşdeğer olduğu gösterilmektedir [12-14].

Buraya kadar ayrık olay sistemlerinin kontrolüne yönelik yapılan tasarımda, kontrol teorisinde olduğu gibi sisteme ilişkin bir model kullanılmaktadır. Bu modelden faydalanılarak tasarım sürecine kontrolör davranışının kilitlenmesiz ve kontroledilebilir olma özelliklerinin incelenmesi de dahil edilmektedir. Ancak ayrık olay sistemlerinin kontrolünde doğrudan kontrolör davranışının tasarlandığı yöntemler yaygın olarak kullanılmaktadır. Daha önce de belirtildiği gibi bu yöntemler kontrolör davranışının bir gösterim biçimi (otomat, Petri ağları gibi) ile, ya

da mantıksal fonksiyonlar kullanarak ifade edilmesine dayanmaktadır. Aşağıda bu yöntemler kısaca özetlenmektedir.

3.2 Modele Dayalı Olmayan Yöntemler

Modele dayalı olmayan yöntemler, genellikle istenilen kontrol davranışının doğrudan modellenmesinden ve gerçekleme aşamasında bu modele karşılık gelen uygulamanın geliştirilmesinden ibarettir. Bu tür yöntemler, sistemin serbest davranışının modeli ile ilgilenmez, doğrudan kontrol davranışı modellenir. Bu yöntemlerde, kilitlenme ya da kontrol edilebilirlik gibi doğrulama amaçlı analizler, sistem modeli kullanılmadığından yapılamamaktadır.

Kontrol davranışının ifadesi farklı şekillerde yapılabilir. Çoğu uygulamada istenen davranış, eşzamanlı (senkron) ya da eşzamansız (asenkron) ardışıl devre yaklaşımı ile önce durum geçiş diyagramlarının, daha sonra da bu diyagramlardan yararlanarak Karnough ya da Quine-McClusey gibi indirgeme yöntemlerinin kullanılarak mantıksal fonksiyonlar şeklinde ifade edilebilir [43]. Bu durumda gerçekleme aşaması bu mantıksal fonksiyonlarla aynı davranışı sergileyecek bir aracın kullanılmasından ibarettir. İleride bahsedileceği gibi kuramsal tasarımın varsayımları ile gerçekleme ortamının kurallarının tam olarak örtüşmemesi bu tür yöntemlerin kullanılmasında da güçlüklere neden olabilmektedir.

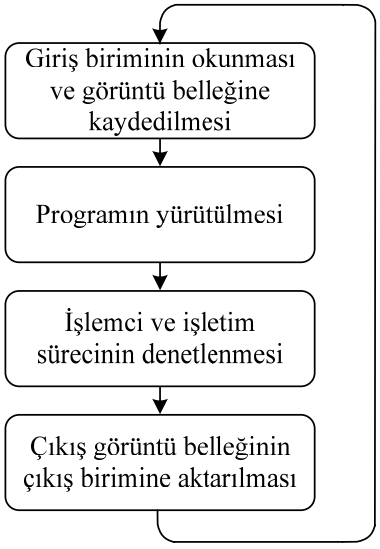
Sonlu durum makineleri ya da otomatlar, sistem modeline dayalı olmayan kontrol yöntemlerinde de kontrol davranışının ifadesi için kullanılmaktadırlar. Bölüm 2'de de bahsedildiği gibi Petri ağları da kontrol uygulamalarında yaygın olarak kullanılmaktadır.

Bu çalışmada, modele dayalı olan ya da olmayan yöntemlerin kullanılmasından bağımsız olarak, sonlu durum makinesi ile ifade edilen bir tasarımın teknolojik araçlar kullanılarak gerçeklenmesine ilişkin yöntemlerle ilgilenilmektedir. Tasarımın gerçekleneceği araç olarak uygulamada çok yaygın kullanılan Programlanabilir Lojik Kontrolörler (PLC) seçilmiştir. İlk kez 1969 yılında röleli elektriksel kumanda devrelerinin yerine kullanmak üzere geliştirilen bu özel sayısal işlemciler, günümüzde seri üretimin yapıldığı hemen her tesiste farklı ölçeklerde uygulama alanı bulmaktadır. Aşağıda PLC'ler çalışma prensipleri ve temel program komutları ile beraber kısaca tanıtılmaktadır. Daha detaylı bilgi için [43,44] kaynakları görülebilir.

3.3 Programlanabilir Lojik Kontrolör (PLC)

Programlanabilir Lojik Kontrolörler (PLC) endüstriyel kumanda ve kontrol uygulamalarında kullanılmak üzere geliştirilmiş mikrokontrolör tabanlı özel sayısal işlemciler olarak tanımlanabilir. İlk PLC'ler elektriksel kumanda devreleri yerine kullanılmak üzere geliştirildiğinden sadece mantıksal işlemleri yerine getirmekteydi. Günümüzde gelişmiş işlem yetenekleri ile PLC'ler sadece mantıksal işlemler için değil, geribeslemeli kontrol uygulamaları için de kullanılmaktadır. Buna rağmen bu aygıtlar için başlangıçta kullanılan PLC ismi kullanılmaya devam etmektedir.

Bir PLC genel anlamda giriş birimi, işlemci birimi ve çıkış birimi olarak üç temel birimden oluşur. Giriş birimi elektriksel işretlerin sayısal verilere dönüştürülmesini, işlemci birimi sistem programını ve kullanıcı programını yürüterek çıkışlara ilişkin sayısal verilerin hesaplanmasını, çıkış birimi ise elde edilen sayısal verilerin elektriksel işaretlere dönüştürülmesini sağlar. PLC'lerde kullanıcı programının işlenmesi bir sistem programı tarafından denetlenir. Sistem programı kullanıcı programının yürütülmesi, kesmeli çalışmanın ve iletişim olaylarının düzenlenmesi ve sistemin çalışma durumlarının denetlenmesi işlevlerini yerine getirir. Kullanıcı programının yürütülmesi Şekil 3.5'te verildiği gibi çevrimsel olarak gerçekleşir. Kesmeli çalışma ya da iletişim işlemlerinin yapılmadığı normal durumlarda bu çevrim sürekli olarak tekrarlanır.



Şekil 3.5 : PLC'de kullanıcı programının çevrimsel olarak yürütülmesi

PLC'lerde kullanıcı programı, bu aygıtlar için geliştirilmiş özel programlama dilleri ile geliştirilmektedir. Kullanılan diller PLC markasına bağlı olarak birtakım

farklılıklar gösterebilmektedir. Günümüzde PLC'lerin programlanmasında metin ya da grafik tabanlı farklı programlama dilleri kullanılmaktadır. IEC 61131-3 standardı tarafından, komut (STL: Statement List), yapısal metin (ST: Structured Text) merdiven mantığı (LAD: Ladder Diagram), fonksiyon blok (FBD: Functional Block Diagram) ve ardışıl fonksiyon gösterimi (SFC: Sequential Function Chart) olmak üzere PLC'ler için beş farklı dil tanımlanmıştır [46]. Bunlardan komut ve merdiven mantığı geleneksel olarak sıklıkla kullanılan programlama dilleridir. Komut ile programlamada mikroişlemcilerin programlanmasında kullanılan Assembly diline benzer düşük seviyeli metin tabanlı bir dil kullanılır. Merdiven Diyagramı ise geleneksel kumanda devrelerinin tasarımında kullanılan gösterime benzer grafik tabanlı bir dildir ve özellikle bakım personeli tarafından takibi ve algılaması kolay olduğu için tercih edilmektedir.

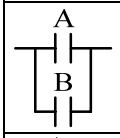
Günümüzde çoğu PLC markasının programlama dilleri IEC 61131-3 standardına uyumludur. Bununla beraber standartta belirlenene benzeyen, ancak kendi programlama dillerini kullanan PLC markaları da mevcuttur. Bu tür dillerdeki farklılıklar, genellikle ilgili PLC markasına özel genişletilmiş komut kümelerine ilişkindir. Bu çalışmada, gerçekleme aşamasında Siemens Simatic S7-200 PLC'ler ve bu PLC'lere ilişkin komut (STL) ve merdiven mantığı (LAD) dilleri kullanılmaktadır [45]. Çizelge 3.1'de S7-200 PLC'ler için temel mantıksal işlem komutları STL dilinde verilmiştir. Mantıksal işlem komutları PLC işlemcisinde bir mantıksal yığın belleği üzerinde işlenir. Çizelge 3.1'de verilen mantıksal işlem komutları, bu yığın belleğe değer atayarak ya da yığın bellekteki değerlere mantıksal işlem uygulayarak ilgili fonksiyonun gerçeklenmesinde kullanılır. Atama komutu ise, yığın belleğin ilk seviyesinde hesaplanan değeri bir mantıksal değişkene atamak için kullanılır.

Merdiven mantığı ile programlamada ise özellikle geleneksel kumanda devrelerinde kullanılan elektriksel gösterim standartlarına çok benzeyen grafik tabanlı bir dil kullanılır. Bu dille, verilen bir mantıksal fonksiyon, mantıksal değişkenlere karşılık gelen kontak gösterimlerinin uygun kombinasyonlarla bağlanmasıyla gerçeklenebilir. Çizelge 3.2'de merdiven mantığında bir mantıksal değişken için kullanılan temel gösterimler verilmektedir.

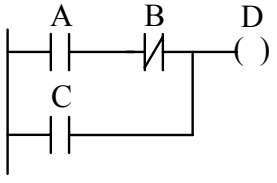
Çizelge 3.1 : S7-200 PLC'ler için temel mantıksal işlem komutları

LD	Yığın belleğe yazma işlemi
A	VE işlemi
O	VEYA işlemi
LDN	Tümleyenini yığına yazma işlemi
AN	Tümleyenine VE işlemi
ON	Tümleyenine VEYA işlemi
ALD	Yığın belleğinin ilk iki seviyesi için VE işlemi
OLD	Yığın belleğin ilk iki se viyesi için VEYA işlemi
=	Atama

Çizelge 3.2 : Merdiven mantığında değişkenler için kullanılan temel gösterimler

	Mantıksal A değişkeni
	Mantıksal A değişkeninin değili
	"A VE B" işlemi
	"A VEYA B" işlemi
	Mantıksal A değişkenine atama

Çizelge 3.1 veya Çizelge 3.2 kullanılarak verilen bir mantıksal fonksiyon STL ya da LAD dilinde gerçeklenebilir. "$\bullet$" mantıksal VE işlemine, "+" mantıksal VEYA işlemine karşılık gelmek üzere Şekil 3.6'da $D = A \bullet \overline{B} + C$ ifadesini gerçekleyen komut ve merdiven mantığı ile oluşturulan program parçaları görülmektedir.




```
LD    A
AN    B
O     C
=     D
```

Şekil 3.6 : $D = A \bullet \overline{B} + C$ ifadesini gerçekleyen LAD ve STL programları

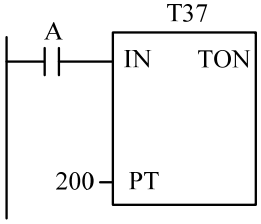
Şekil 3.6'da görülen LAD programında mantıksal bir fonksiyon hesaplanmakta ve bir değişkene atama yapılmaktadır. Bu şekilde hesap ve atamanın yapıldığı, bütünlük taşıyan program parçaları, S7-200 PLC'lerin programlama editöründe "Network" adını alır. Bu çalışmada merdiven mantığında bütünlük taşıyan bu tür program parçalarına "basamak" adı verilecektir. Bir PLC programında genel durumda çok

sayıda basamak bulunabilir. Şekil 3.5'te verilen çalışma prensibine göre kullanıcı programı ilk basamaktan son basamağa doğru sırayla işlenerek çıkışlar hesaplanır.

Uygulamada bir kontrol davranışının oluşması için belirli bir durumda belirli bir sürenin geçmesi gerekebilir. Ya da kontrol davranışının gerçekleşmesi bir olayın belirli bir sayıda olmasına bağlı olabilir. PLC'lerde sırasıyla zamanlama ve sayma olarak adlandırılan ve bu davranış türlerinin gerçekleştirilmesi için hazır program blokları mevcuttur. Bu program blokları sırasıyla zamanlayıcılar ve sayıcılar olarak adlandırılır. Zamanlayıcıların gecikmeli kapatan (on delay-TON), kalıcı gecikmeli kapatan (on delay retantive-TONR) ve gecikmeli açan (off delay-TOF) gibi türleri bulunmaktadır. Sayıcıların ise sayma yönüne göre ileri (CTU), geri (CTD) ve ileri-geri (CTUD) gibi türleri bulunmaktadır. Aşağıda genel zamanlama işlemleri için kullanılabilecek gecikmeli kapatan zamanlayıcının ve genel sayma işlemleri için kullanılabilecek ileri sayıcının çalışma prensipleri anlatılmaktadır.

3.3.1 Gecikmeli kapatan zamanlayıcı (TON)

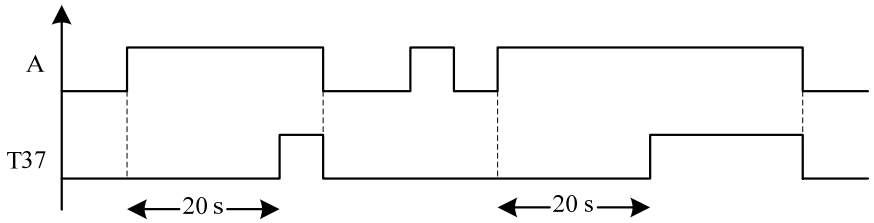
Gecikmeli kapatan sayıcılar girişine uygulanan bir işaret ile tanımlanan bir süre sonunda çıkış üretir, giriş işareti kesildiğinde çıkış sıfırlanır. Şekil 3.7' de S7-200 PLC'lerinde kullanılan TON zamanlayıcının LAD ve STL dilindeki kullanım örneği verilmektedir. Bu PLC'lerde zamanlayıcılar için T37 gibi geçerli bir zamanlayıcı numarası kullanılır. Zamanlamayı başlatan işaret "IN" girişine uygulanır. "PT" girişi ise gecikme süresine karşılık gelmektedir. S7-200 PLC'lerde PT girişi tam sayı değerleri alır ve gerçeklemede kullanılan zamanlayıcı numarasına bağlı olarak, t saniye cinsinden istenen gecikme süresi olmak üzere $10\times t$, $100\times t$, $1000\times t$ değerlerini alabilir. T37-T63, T101-T255 zamanlayıcıları için bu değer $10\times t$ 'dir. Buna göre, Şekil 3.7'de PT girişine atanan 200 değeri ile T37 zamanlayıcısı 20 s.'lik gecikme süresine kurulmuştur.




```
LD      A
TON     T37, 200
```

Şekil 3.7 : TON zamanlayıcısının LAD ve STL dillerindeki kullanımı

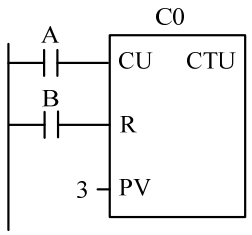
Şekil 3.7'deki program parçasına göre A mantıksal değişkeninin değeri 1 olduğunda T37 zamanlayıcısı süreyi başlatır ve 20 s. sonra çıkış verir, yani T37 mantıksal 1 değerini alır. Şekil 3.8'de bu örneğin çalışmasına ilişkin zaman diyagramı verilmiştir.



Şekil 3.8 : Örnek TON zamanlayıcısına ilişkin zaman diyagramı

3.3.2 İleri sayıcı (CTU)

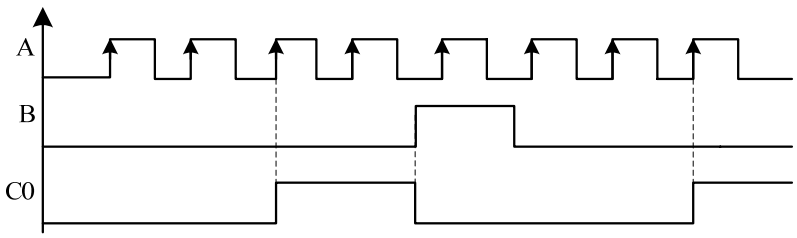
İleri sayıcı sayma girişine uygulanan işaretin yükselen kenarı ile (mantıksal 0'dan 1'e geçişi ile) içeriğini bir artırır, silme girişine uygulanan işaretle içeriğini sıfırlar. Şekil 3.9'da S7-200 PLC'lerde kullanılan CTU sayıcısı görülmektedir. S7-200 PLC'lerde sayıcılar C0'dan C255'e kadar 256 adet sayıcı bloğu kullanılabilir. Şekil 3.9'da görülen sayıcının sayma girişi "CU", silme girişi "R" dir. "PV" girişi sayım için hedef değerdir ve bu değere erişilmesi ile sayıcı çıkış verir. Bu örnekte A işaretinin 0'dan 1'e geçişleri sayılmaktadır. A işaretinin 3. kez mantıksal 1 değeri aldığı an C0 sayıcısı çıkış verir, yani C0 değişkeni mantıksal 1 değerini alır. B değişkeninin mantıksal 1 değeri almasına kadar bu değerini korur.




```
LD      A
LD      B
CTU     C0, 3
```

Şekil 3.9 : CTU sayıcısının LAD ve STL dillerindeki kullanımı

Şekil 3.10'da bu sayıcı örneğinin çalışmasına ilişkin zaman diyagramı verilmiştir.



Şekil 3.10 : Örnek CTU sayıcısına ilişkin zaman diyagramı

4. ZS-OTOMAT

Genel olarak bir sisteme istenilen bir kontrol davranışın kazandırılması, davranışa karşılık gelen kontrol kurallarına ilişkin tasarımı ve bu kuralları uygun bir teknoloji ile hayata geçirilmesini, yani gerçekleme aşamalarını içerir. Ayrık olay sistemlerinin kontrolünde de tasarım sonucu istenilen AOS davranışına karşılık gelen kontrol kuralları belirlenir ve bu kurallar uygun bir modelleme (gösterim) biçimi ile ifade edilir. Gerçeklenme aşamasında ise bu modelleme biçiminin ifade kurallarının kullanılan teknolojiye uygun bir biçimde hayata geçirilebildiği yöntemlere ihtiyaç duyulur.

Günümüzde, Bölüm 2'de bahsedildiği gibi formal tasarım aşamasında sonlu durum makineleri ya da otomat gösterim biçimi yaygın olarak kullanılmaktadır. Bu bölümde AOS kontrolünde sıklıkla karşılaşılan, fakat bu gösterim biçimi ile ifade edilemeyen ya da ifadesi zor olan iki tür AOS davranışından bahsedilecek, bu davranış türlerini tanımında bulunduran bir otomat yapısı önerilecektir. Önerilen yapı genel olarak ele alınan bir AOS'nin modellenmesi için kullanılabilecektir, ancak burada öncelikle amaçlanan, AOS kontrolünde tasarım sonucu elde edilen kontrolör ya da üst denetleyicilerin modellenmesidir. Tanımlanan otomat yapısı Bölüm 5'te tanıtılacak olan gerçekleme yöntemi ile uyumlu olacaktır.

4.1 Zamanlama ve Sayma Davranışları

Ayrık Olay Sistemlerinin davranışlarının ifadesinde kullanılan ve Bölüm 2.4'te bahsedilen otomat gösterim biçimi ile uygulamada sıklıkla karşılaşılan iki tür davranışın ifadesi oldukça güçtür. Bunlardan "zamanlama" olarak adlandırılacak davranış türü uygulamada belirli bir durumda yeni bir olayın oluşması için geçmesi gereken süre ile ilgilidir. Diğer davranış ise "sayma" olarak adlandırılacaktır ve yeni bir duruma geçişin belirli bir durumda, belirli bir olayın, belirli bir sayıda olması koşuluna karşılık gelir.

Karmaşıklığa neden olmaması açısından Bölüm 2.4'te tanıtılan otomat gösterimi bundan sonra "standart otomat" olarak anılacaktır.

Zamanlama davranışı kontrol uygulamalarında, farklı gereksinimlerle ortaya çıkabilir. Trafik ışıklarının belirli sürelerde yanıp sönmesi, endüstriyel üretim süreçlerinde bir işlemin başlaması için başka bir işlemin bitmesinden sonra belirli bir sürenin geçmesi, ya da bir test işleminin sonucuna karar vermek için belirli bir süre beklenmesi gibi örnekler verilebilir. Sayma davranışına ise bir kontrol davranışının belirli bir durumda, belirli bir olayın, belirli bir sayıda olması koşulu ile oluştuğu uygulamalarda gereksinim duyulur. Bir paketleme ünitesinde üretilen her belirli sayıdaki ürün için bir paket oluşturulması örnek olarak verilebilir.

Zamanlama ve sayma davranışlarını standart otomat gösteriminde doğrudan ifade edecek bir yapı bulunmamaktadır. Ancak, uygulamada standart otomat gösterimi ile ifade edilen bir tasarım için zamanlama ve sayma davranışlarını gerçeklemek imkânsız değildir. Örneğin zamanlama davranışının gerçeklenmesi problemi için, öngörülen bekleme sürelerinin dolması birer olay olarak tanımlanabilir ve kontrolör modeli bu olayları içerecek şekilde oluşturulabilir. Ancak gerçekleme aşamasında bu sürelerin dolması ile ilgili olayı üretecek bir mekanizmanın da bulunması gerekir. Bu durumda zamanlama işlemi formal kontrolör modeli içinde değil, gerçekleme için kullanılan yöntemlerle ifade edilmiş olur. Sayma davranışını ifade etmek içinse standart otomat yapısında bir olay n kez sayılacaksa durumlar arası geçişlerin bu olayla sağlandığı n adet durum sayma davranışını karşılamak için kullanılabilir. Ancak bu en genel halde durum sayısı çok fazla olan otomatların elde edilmesine neden olabilir.

Zamanlama ve sayma davranışları için farklı otomat yapılarının kullanılması öngörülebilir. Örneğin, Bölüm 2.7'de tanıtılan zamanlı otomatın [36,40], yapısında olayların oluş zamanları ile ilgili bilgiyi bulundurması nedeniyle zamanlama işlemlerinin tasarımında kullanılması düşünülebilir. Zamanlı otomat, ayrık olay sistemlerinde zamanlı dil modeli için kullanılan bir gösterim biçimidir. Yani, olay dizileri zaman bilgileri ile beraber ifade edilir. Ayrık olay sistemlerinin kontrolü için yapılan tasarımda, tasarımcının zamanlama davranışını olaylardan çok, durumlar için öngördüğü söylenebilir. Zamanlama davranışının durumlar esas alınarak atanması, zamanlama davranışına ilişkin tasarımın, ilgili durumlara hangi olayla gelindiğinden bağımsız olarak yapılmasını sağlar. Zamanlama davranışının durumlarla

ilişkilendirilmesi gerçekleme için de önemli kolaylıklar getirmektedir. Gerçeklemede, zamanlama işlevinin hangi koşul süresince yerine getirileceği bilgisi durumlarla doğrudan temsil edilebilir. Böyle bir bilginin anlık olaylarla doğrudan ifadesi mümkün değildir. Özel olarak PLC'ler ile gerçeklemede Bölüm 3.3.1 ve Şekil 3.8'de gösterildiği gibi zamanlayıcı, girişine uygulanan sürekli bir işaret ile çalışmaktadır. Gerçekleme aşamasında bu sürekli işaret için, durumu temsil eden bir değişkenden yararlanılabilir.

Bölüm 2.8'de tanıtılan PLC-otomat yapısı [41], yukarıda yapılan irdelemeye paralel olarak, zamanlama davranışını durumlar üzerinden belirlemektedir. Bununla beraber, durum geçişleri olaylarla değil, giriş olarak adlandırılan ve anlık olmayan işaretlerle ilişkilendirilmiştir. Bu anlamda PLC-otomatta durumlara atanan süreler, her durum için ayrı belirlenen bir giriş kümesi için bir çeşit gecikme işlemini yerine getirmektedir. Belirli bir duruma ilişkin geçişler, o durumda gecikme süresi dolduktan sonra ilgili girişler hala mevcutsa gerçekleşebilir. Süre dolmadıkça o durum ilgili işaretlere duyarsızdır. Durumlar için belirlenen işaret kümelerinin dışındaki işaretlere ise böyle bir gecikme uygulanmaz.

Görüldüğü gibi PLC-otomatta zaman gecikmesi, durumlar aracılığı ile durum geçişlerine neden olan ve anlık olmayan giriş işaretlerine uygulanmaktadır. Ayrık olay sistemleri yaklaşımında durum geçişlerine bu türden bir gecikmenin uygulanması mümkün değildir, çünkü durum geçişleri anlık olaylarla ilişkilendirilmiştir. Bunun dışında her durum için gecikme ile ilgili giriş kümelerinin ayrı ayrı tanımlanması, gerçekleme aşamasında ilave güçlüklere neden olacaktır.

Sayma davranışının ise yukarıda bahsedilen otomat yapıları ile doğrudan, az sayıda durumla ifade edilmesi mümkün değildir.

Uygulamada zamanlama ve sayma davranışının gerçeklenmesinde kullanılan teknolojik aracın özelliklerinden yararlanılmaktadır. Bölüm 3'te kısaca tanıtıldığı gibi Programlanabilir Lojik Kontrolörler (PLC) yapılarında zamanlayıcı ve sayıcı adı verilen hazır program bloklarını içermektedir. Tasarımda formal yöntemlerin kullanılıp kullanılmamasından bağımsız olarak günümüzün AOS kontrol uygulamalarında PLC'ler yaygın olarak kullanılmakta ve zamanlama ve sayma davranışları bu hazır bloklarla gerçeklenmektedir.

Bu bölümde, tasarlanmış kontrolör ya da üst denetleyiciyi gerçekleme amaçlı olarak modellemek üzere zamanlama ve sayma davranışlarını yukarıda bahsedilen zorluklar olmaksızın yapısında barındıran, teknolojik elemanların sağladığı fonksiyonları doğrudan kullanmayı mümkün kılan bir otomat yapısı elde edilecektir.

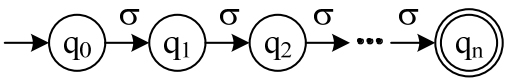
Zamanlama ve sayma davranışlarının yanında, kontrolör ya da üst denetleyici tarafından üretilmesi gereken olay ya da işaretlerin de ifadesi önem taşır. Uygulamada standart otomat tanımında bulunmamasına rağmen, kontrolör ya da üst denetleyicilerin sonlu durum makineleri ile ifadesinde olay çıkışları olan yapılar kullanılmaktadır [19,23,32]. Olay çıkışlarının üretilmesi farklı koşullara bağlı olabilir. Belirli bir kontrol davranışına karşılık gelen çıkışın, belirli bir duruma geçilmesi, bir durumda belirli süre beklenmesi ya da bir olayın belirli bir sayıda olmasına bağlı olarak üretilmesi gerekebilir. Sonuç olarak olay çıkışları otomatın bulunduğu duruma ya da durumla beraber zamanlama ve sayma davranışlarıyla doğrudan ilgilidir. Bu bölümde elde edilecek otomat tanımında, durumlara ve zamanlama-sayma özelliklerine bağlı olarak olay üreten çıkış fonksiyonları da bulundurmaktadır.

4.2 Zamanlama ve Sayma Davranışlarının İfadesi

Yukarıda da bahsedildiği gibi standart otomat ile zamanlama ve sayma davranışlarının gerçeklenmesi imkânsız değildir. Belirli bir durumda belirli bir sürenin dolduğu bilgisi "süre doldu" olayı üreten bir mekanizma kullanılarak modellenebilir. Başka bir yöntem ise bir saat işareti kullanmak ve saat işaretinin saymak olabilir. Saat işaretinin yükselen kenarı bir olaya karşılık getirilirse bu olayın belirli bir sayıda olması, öngörülen sürenin dolması anlamına gelecektir. Bu şekilde zamanlama davranışının ifadesi sayma davranışına indirgenmiş olur. Bu nedenle aşağıda sayma davranışının ifadesi incelenecek, elde edilen sonuçlar zamanlama davranışı için de kullanılacaktır.

Aşağıda gösterileceği üzere genel durumda sayma davranışının standart otomat gösterimi ile ifadesi kabul edilemeyecek kadar çok sayıda durum kullanılmasına neden olabilir. Bu nedenle temel amaç sayma davranışını çoklu durum yerine az sayıda durumla ifade edecek bir yapı elde etmek olacaktır. Sonuçlar zamanlama davranışı için de kullanılacaktır. Öncelikle sayma işleminin standart otomat gösteriminden yola çıkılarak ifadesi elde edilecektir.

Şekil 4.1'deki gibi bir σ olayının art arda *n* kez oluşunu ayırt edecek bir standart otomat modelini ele alalım. Bu otomatın q_n işaretli durumuna erişmesi ile q_0 durumundan itibaren *n* kez arka arkaya oluşan bir σ olayı belirlenmiş olur.



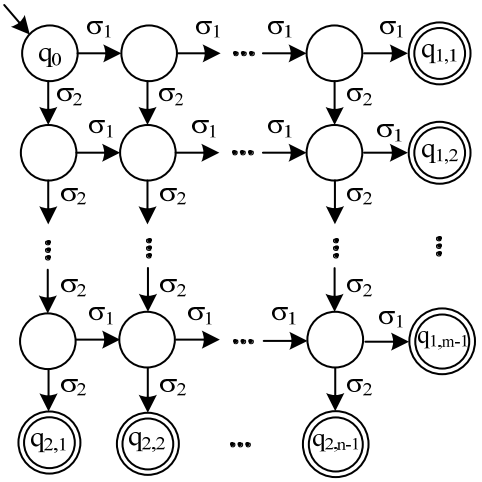
Şekil 4.1 : σ olayının *n* kez sayılmasını modelleyen standart otomat gösterimi

Şekil 4.1' deki durum geçişi kapalı olarak, uzunluğu *n* olan, $s = \sigma\sigma...\sigma$ şeklinde bir kelime ile

$$f(q_0, s) = q_n \qquad \textbf{(4.1)}$$

şeklinde ifade edilebilir.

Belirli bir durumdan itibaren birden fazla sayıda olayın sayılması ise durum sayısının çarpımsal olarak büyümesini gerektirecektir. Şekil 4.2' de q_0 durumundan itibaren σ_1 olayını *n* kez, σ_2 olayını ise *m* kez sayan bir standart otomat gösterimi verilmiştir.

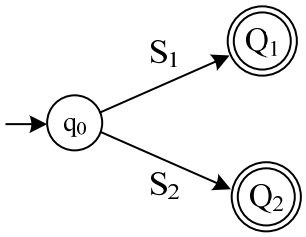


Şekil 4.2 : σ_1 ve σ_2 olaylarının sayılmasını modelleyen standart otomat gösterimi

Sayma davranışının az sayıda durumla, hatta mümkünse tek bir durumla ifadesi, Standart otomat gösterimine yapısal yenilikler getirilerek elde edilen bir gösterilimle sağlanabilir. Şekil 4.2' de görüldüğü gibi hangi olayın sayısının tamamlandığı iki grup durumdan birine erişilmesi ile belirlenmektedir. $i = 1,2,...,m-1$ olmak üzere $q_{1,i}$ durumları birinci gruba, $i = 1,2,...,n-1$ olmak üzere $q_{2,i}$ durumları ise diğer gruba karşılık gelmektedir. Otomat bir gruptaki işaretli durumlardan birine eriştiğinde daha sonraki davranışın ne olacağı grubun hangi elemanına eriştiği ile

ilgili değildir. Çünkü sayma davranışı bir olayın belirlenen sayıda meydana gelip gelmediğini belirlemektedir ve bu anlamda, bir gruptaki durumların her biri eşdeğerdir. Bu eşdeğerlik sayma davranışını az sayıda durumla ifade etmek için kullanılabilinir.

Şekil 4.2'deki otomatın işaretlediği dil bu açıdan iki farklı kümeye ayrılabilir. $q_{1,i}$ işaretli durumlarının oluşturduğu küme Q_1 ile, $q_{2,i}$ işaretli durumlarının oluşturduğu küme de Q_2 ile gösterilsin. Q_1 durum kümesinin elemanları ile sonlanan tüm olay dizileri yani kelimeler S_1 kümesinin, Q_2 grubuyla sonlanan tüm kelimeler de S_2 kümesinin elemanlarını oluşturtsun. Buna göre Şekil 4.2'deki sayma davranışına eşdeğer bir davranışa sahip olan sembolik bir gösterim Şekil 4.3'teki gibi verilebilir.



Şekil 4.3 : İki farklı olayın sayıldığı davranış için eşdeğer sembolik bir gösterim.

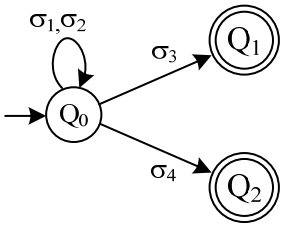
Şekil 4.3'teki sembolik gösterime ilişkin durum geçişleri $s_1 \in S_1$, $s_2 \in S_2$, $q_{1,i} \in Q_1$ ve $q_{2,i} \in Q_2$ olmak üzere kapalı olarak şu şekilde ifade edilebilir:

$$\begin{aligned} f(q_0, s_1) &= q_{1,i} \\ f(q_0, s_2) &= q_{2,i} \end{aligned} \qquad \textbf{(4.2)}$$

Şekil 4.3 ve Eşitlik 4.2 ilerleyen satırlarda tanımlanacak olan zamanlama ve sayma özelliği bulunan otomat gösterimi için temel çıkış noktası olacaktır. Standart otomata eklenecek yeni yapısal özelliklerle zamanlama ve sayma davranışı Şekil 4.3'teki sembolik gösterime paralel olarak az sayıda durumla ifade edilecektir. Elde edilecek bu otomat ifadesinde temel amaç tasarlanmış kontrol davranışının gerçeklenmesine yönelik olmasıdır. Sonuçta, gerçeklemede kullanılacak teknolojik araçların mümkün kıldığı zamanlama ve sayma fonksiyonlarının doğrudan kullanılmasına uygun bir yapı önerilecektir.

Otomatların durum diyagramları ile gösteriminden alışılageldiği üzere, durum geçişleri, geçişe neden olan olayı temsil eden yönlü bir ok ile ifade edilir. Bu

bölümde tanımlanacak olan durum diyagramı gösteriminde de Şekil 4.3'teki sembolik gösterimden farklı olarak, durum geçişleri kelimelere değil, olaylara ilişkilendirilecektir. Bunu sağlamak üzere Şekil 4.2'de davranış açısından eşdeğer olan ve ilk durum q_0'ı da içeren başka bir durum kümesi daha oluşturulabilir. Gerçekten de Şekil 4.2'de işaretli olmayan durumların tamamı sayma işleminin devam ettiği eşdeğer tek bir durum olarak değerlendirilebilir. Sayma işlemi devam ettiği sürece σ_1 ve σ_2 olayları bu eşdeğer durumdan çıkışa neden olmayacağından bu olaylar özçevrim olayları olarak değerlendirilecektir. Başlangıç durumunun içinde bulunduğu eşdeğer durum Q_0 ile gösterilirse Şekil 4.3'te kelimelerin kullanıldığı sembolik gösterim Şekil 4.4'teki gibi bir standart otomat gösterimine karşılık getirilebilir. Görüldüğü gibi sayımı yapılan σ_1 ve σ_2, Q_0 durumunda özçevrim olayları olarak belirmektedir. Durum geçişlerine neden olan ve ilk defa Şekil 4.4'te kullanılan σ_3 ve σ_4 olayları ise sırasıyla σ_1 ve σ_2 olaylarının sayımının tamamlanmasına karşılık gelmektedir. Bu noktada σ_3 ve σ_4 olaylarının, ilgili sayma işlemi tamamlandığında otomatik olarak "üretildiği" varsayılacaktır ve ilerleyen satırlarda *sayma olayı* olarak adlandırılacaktır. Açıktır ki, standart otomat gösteriminde sayma özelliği ve sayıma karşılık bir olayın üretilmesi gibi bir işlev bulunmamaktadır. Bölüm 4.3' te otomat gösterimine bu özelliği kazandıracak bir yapı ile beraber, olayların üretilmesini mümkün kılan çıkış fonksiyonları tanımlanacaktır. Benzer bir özellik zamanlama davranışı için de geliştirilecektir. Bu şekilde uygulamaya yönelik olarak kullanılacak yeni bir otomat tanımı elde edilecektir.



Şekil 4.4 : Sayma davranışına ilişkin eşdeğer otomat

Daha önce de belirtildiği gibi zamanlama davranışı bir saat işaretinin sayılması ile sayma davranışına indirgenebildiğinden Şekil 4.4'te elde edilen sonuçlar doğrudan zamanlama davranışı için kullanılabilir. Ancak Bölüm 4.3'te tanımı yapılacak otomat gösteriminde, zamanlama ve sayma davranışlarını ayırt etmek üzere farklı

terminolojiler kullanılacaktır. Bu otomat, hangi durumda hangi olayların kaç kez sayılması gerektiği bilgisini *Sayma Yapısı* adı verilen bir tanımla taşıyacaktır. Benzer şekilde hangi durumda ne kadar süre beklenmesi gerektiği bilgisi de *Zamanlama Yapısı* adlı bir tanımla verilecektir. Tanımında zamanlama ve sayma davranışlarını bulunduran bu otomat "ZS-yapılı otomat" ya da kısaca "ZS-otomat" olarak anılacaktır.

Görüldüğü gibi Şekil 4.4'teki otomata Sayma Yapısı'nın eklenmesi ile, Şekil 4.2 ile verilen ve farklı iki olayın sayılmasını modelleyen standart otomat gösterimine eşdeğer bir davranış elde edilmektedir. Q_0, Q_1 ve Q_2 durum kümelerinin, küme yerine tekil durumlar olarak değerlendirilmesi, otomat davranışının ifade edilmesi açısından bir farklılık oluşturmamaktadır. Bu nedenle, sayma ve zamanlama davranışlarının ifadesi durum kümeleri yerine durumlar üzerinden yapılacaktır.

Aşağıda ZS-otomatın formal tanımda kullanılacak *vektör ağırlığı* ve *ağırlık kaybı* tanımları verilmiştir.

Tanım 4.1 Bir **u** vektörünün *ağırlığı* vektörün sıfırdan farklı elemanlarının sayısına eşittir ve AG(**u**) ile gösterilir. ■

Tanım 4.2 $j,k \in \mathbf{N}$ olmak üzere her adımda sadece bir elemanının değeri değişen bir $(\mathbf{u}^1,\mathbf{u}^2,...,\mathbf{u}^k,...)$ vektör dizisinin k. adımında j. elemanının aldığı değer $u^k(j)$ ile gösterilsin. Bu vektör dizisinin j. elemanını için (k-1) ve k adımlarında

$$\begin{aligned} u^{k-1}(j) &\neq 0 \\ u^k(j) &= 0 \end{aligned} \qquad \textbf{(4.3)}$$

koşulları sağlanıyorsa (k–1). adımdan k. adımına geçişte **u** vektörünün ağırlığı bir azalır, yani

$$AG(\mathbf{u}^{k-1}) - AG(\mathbf{u}^k) = 1 \qquad \textbf{(4.4)}$$

koşulu sağlanır. Bu durumda k adımı *ağırlık kayıp adımı*, j tam sayısı da k adımındaki *kayıp eleman* olarak adlandırılır ve

$$AG_{\downarrow}(\mathbf{u}^k) = j \qquad \textbf{(4.5)}$$

ile gösterilir. Ağırlık kaybının olmadığı adımlar için kayıp eleman tanımsızdır. ■

4.3 ZS-otomat

ZS-otomat şu şekilde tanımlanmaktadır:

$$G_{ZS} = (Q, \Sigma, f, \Gamma, T, C, \Delta, q_0) \quad \textbf{(4.6)}$$

Burada,

- Q sonlu durumlar kümesi,
- Σ sonlu olaylar kümesidir. $\Sigma = \Sigma_S \cup \Sigma_T \cup \Sigma_C$ şeklinde üç alt kümenin bileşimi olarak ifade edilir. $\Sigma_S = \{\sigma_1, \sigma_1, ..., \sigma_n\}$ *sistem olayları* kümesidir. Sistem olayları Standart otomat tanımı ile verilen olaylar kümesindeki olaylara eşdeğerdir. $\Sigma_T = \{\sigma^T\}$, tek elemanlı bir kümedir ve elemanı *zamanlama olayı*dır. $\Sigma_C = \{\sigma_1^C, \sigma_2^C, ..., \sigma_n^C\}$ *sayma olayları* kümesidir. Bir ZS-otomatta sistem olayları sayısı kadar sayma olayı bulunur.
- $f : Q \times \Sigma \to Q$ durum geçiş fonksiyonudur ve özellikleri Standart otomat tanımında verildiği gibidir.
- $\Gamma : Q \to 2^Q$ aktif olay fonksiyonu adını alır ve özellikleri Standart otomat tanımında verildiği gibidir.
- T Zamanlama Yapısı (ZY) olarak adlandırılır ve

  $$T = \{\tau_q, \ q \in Q\} \quad \textbf{(4.7)}$$

  şeklinde tanımlanır. Burada $\tau_q \in \mathbf{R}_{\geq 0}$, ZS-otomatın q durumunda üretilecek zamanlama olayına ilişkin gecikmeyi tanımlar.
- C Sayma Yapısı (SY) olarak adlandırılır ve

  $$C = \{\mathbf{c_q}, \ q \in Q\} \quad \textbf{(4.8)}$$

  şeklinde tanımlanır. $\Sigma_S = \{\sigma_1, \sigma_1, ..., \sigma_n\}$ sistem olayları kümesi olmak üzere, $\mathbf{c_q}$ q durumuna ilişkin *n* boyutlu doğal sayılar vektörüdür. $\mathbf{c_q}$ vektörünün $\sigma_j \in \Sigma_S$ olayına karşılık gelen $c_q(j)$ elemanı, q durumunda bu olayının kaç kez sayılacağını belirlemektedir.
- Δ, Çıkış Fonksiyonları kümesidir ve

  $$\Delta = \{\delta, \delta^T, \delta^C\} \quad \textbf{(4.9)}$$

  şeklinde üç bileşenle verilir. Bunlardan $\delta : Q \to \Sigma_S \cup \{\varepsilon\}$, *duruma bağlı çıkış fonksiyonu*dur ve $q \in Q$ için,

$$\delta(q) = \begin{cases} \varepsilon & \text{q durumu aktif degil} \\ \exists!\sigma \in \{\Sigma_S \cup \{\varepsilon\}\} & \text{q durumu aktif} \end{cases} \qquad \textbf{(4.10)}$$

şeklinde tanımlanır. Burada ε boş kelime, ya da uzunluğu sıfır olan kelimedir.

$\delta^T : Q \times \mathbf{R}_{\geq 0} \rightarrow \Sigma_T \cup \{\varepsilon\}$ *duruma ve ZY'ye bağlı çıkış fonksiyonu*dur ve

$$\delta^T(q,t) = \begin{cases} \varepsilon & \text{q durumu aktif degil veya } \tau_q = 0 \\ \varepsilon & \text{q durumu aktif ve } t \neq t_q + \tau_q \\ \sigma^T \in \Sigma_T & \text{q durumu aktif ve } t = t_q + \tau_q \end{cases} \qquad \textbf{(4.11)}$$

şeklinde tanımlanır. Burada t zamana, t_q ise q durumunun aktif olduğu ana karşılık gelir. $\tau_q \in T$ 'dir.

$\delta^C : Q \times \mathbf{N} \rightarrow \Sigma_C \cup \{\varepsilon\}$ *duruma ve SY'ye bağlı çıkış fonksiyonu* adını alır. δ_q^C, Tanım 4.1 ve Tanım 4.2 kullanılarak,

$$\delta^C(q,k) = \begin{cases} \varepsilon & q \text{ durumu aktif degil veya } \mathbf{c}_q = \mathbf{0} \\ \varepsilon & q \text{ durumu aktif ve } AG(\mathbf{c}_q - \mathbf{n}_q^k) = AG(\mathbf{c}_q) \\ \sigma_j^C \in \Sigma_C & q \text{ durumu aktif ve } AG_{\downarrow}(\mathbf{c}_q - \mathbf{n}_q^k) = j \end{cases} \qquad \textbf{(4.12)}$$

şeklinde tanımlanır. $\mathbf{c}_q \in C$'dir. *Sayma adımı* olarak adlandırılan k değişkeni q durumu aktif olduğu an sıfır değerini alır ve durum değişikliğine neden olmayan her olayda (öz çevrim olaylarının meydana gelmesiyle) içeriği bir artar. $\mathbf{n}_q^k$ vektörünün j. elemanı $n_q^k(j)$, $\Sigma_S = \{\sigma_1, \sigma_1, ..., \sigma_n\}$ sistem olayları kümesindeki bir σ_j olayına karşılık gelir. $\mathbf{n}_q^k$, elemanları üzerinden aşağıdaki şekilde tanımlanır:

$$\begin{aligned} & n_q^0(j) = 0, \\ & n_q^{k+1}(j) = \begin{cases} n_q^k(j) + 1 & f(q,\sigma^k) = q, \ \sigma^k = \sigma_j \\ n_q^k(j) & f(q,\sigma^k) = q, \ \sigma^k \neq \sigma_j \end{cases} \qquad k \in \mathbf{N} \end{aligned} \qquad \textbf{(4.13)}$$

Burada σ^k, k. adımdan (k+1). adıma geçişe neden olan olaydır.

■ q_0 başlangıç durumudur. ■

Görüldüğü gibi ZS-otomat, standart otomatın özelliklerine ek olarak zamanlama, sayma ve olay üretme özelliklerine de sahiptir. Bu anlamda standart otomatın, burada tanımlanan ZS-otomatın özel bir durumu olduğu söylenebilir. Gerçekten de (4.7), (4.8) ve (4.10) ifadelerinde $\forall q \in Q$ için $\tau_q = 0$, $\mathbf{c}_q = \mathbf{0}$ ve $\delta(q) = \varepsilon$ sağlanıyorsa ZS-otomat davranışı, standart otomat davranışı ile aynı olacaktır.

f durum geçiş fonksiyonu tanım kümesindeki olay kümelerinin sistem olaylarına, zamanlama olayına ve sayma olaylarına indirgenmesi ile aşağıdaki gibi f_S, f_T ve f_C olarak üç alt kümeye ayrılabilir:

$$f_S \subset f,\ f_S : Q \times \Sigma_S \to Q \quad \textbf{(4.14)}$$

$$f_T \subset f,\ f_T : Q \times \Sigma_T \to Q \quad \textbf{(4.15)}$$

$$f_C \subset f,\ f_C : Q \times \Sigma_C \to Q \quad \textbf{(4.16)}$$

Buna göre,

$$f = f_S \cup f_T \cup f_C \quad \textbf{(4.17)}$$

ilişkisi sağlanır.

ZS-otomatta, zamanlama ve sayma yapıları durumlara ilişkilendirilmiş olarak tanımlanmaktadır. Benzer şekilde çıkış fonksiyonları da durumlarla ilişkilendirilmiştir. Bu, tasarımda ve gerçeklemede büyük esneklik sağlamaktadır. Tasarım aşamasında zamanlama davranışı, olaylardan bağımsız olarak belirli durumlar için öngörülür ve ZY ile bu durumlara belirli gecikme süreleri atanır. ZY'ye bağlı çıkış fonksiyonu ise ZY tarafından gecikme atanan durumlarda, gecikme süresi sonunda zamanlama olayının üretileceğini garantilemektedir. Benzer durum sayma davranışının modellenmesi için de geçerlidir. Belirli olayların sayılması gerektiği durumlara gelindiğinde, hangi olay ya da olayların o durumda kaç kez sayılacağı SY tarafından belirlenir. SY'ye bağlı çıkış fonksiyonu ise ilgili olayın sayımı tamamlandığında sayımın tamamlanmasına karşılık gelen sayma olayının üretilmesini garantilemektedir. Tasarımcı, gecikme süresinin, ya da ilgili sayımın tamamlanmasından sonraki kontrol davranışının ne olacağını, zamanlama ve sayma olaylarını kullanarak kolaylıkla belirleyebilir.

ZY ve SY ile bir duruma aynı anda gecikme süresi ve olay sayma özelliği atanabilir. Bir durumda birden fazla olayın sayılması da mümkündür. Böyle bir tasarımda sayımı ilk tamamlanan olay belirlenmiş olacaktır.

Görüldüğü gibi tasarım aşamasında ZY ve SY ile durumlara bir tür "özellik" kazandırılarak zamanlama ve sayma davranışlarının modellenmesi kolaylaştırılmaktadır. Zamanlama ve sayma davranışlarının durumlarla ilişkili olarak verilmesi gerçekleme aşaması için de avantaj sağlamaktadır. Programlanabilir Lojik Kontrolörlerde zamanlayıcı ve sayıcı program bloklarının çalışma koşullarının

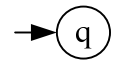
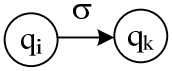
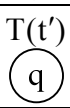
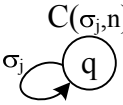
durumlarla ilişkilendirilmesi hem bu davranışların gerçeklenmesini, hem de PLC programının okunabilirliğini kolaylaştırmaktadır.

Bir ZS-otomat (4.6) ifadesindeki bileşenlerinin verilmesiyle tam olarak tanımlanmış olur. Ancak standart otomatlarda olduğu gibi, tasarımda durum diyagramlarını temel alan bir gösterimine gereksinim duyulur. Aşağıdaki bölümde ZS-otomat için durum diyagramı gösterimine temel olacak kurallar tanıtılmaktadır.

4.4 ZS-otomat İçin Durum Diyagramı Gösterilimi

ZS-otomat, standart otomatın bileşenlerine ilave olarak ZY, SY ve çıkış fonksiyonu bileşenlerine sahiptir. Bu nedenle ZS-otomat tanımıyla gelen ZY, SY ve çıkış fonksiyonları için de bir ifade şekli belirlenmelidir. Çizelge 4.1 ZS-otomatın durum diyagramı gösterimindeki yapıtaşlarını ve bunların tanımında verilen karşılıklarını göstermektedir.

Çizelge 4.1 : Durum diyagramı gösteriminde ZS-otomatın yapıtaşları

Bileşen	**Durum Diyagramı Gösterimi**	**Karşılığı**
İlk durum		$q_0 = q$
Durum geçiş fonksiyonu		$f(q_i, \sigma) = q_k$
Duruma bağlı çıkış fonksiyonu		$\delta(q) = \sigma$
ZY' ye bağlı çıkış fonksiyonu		$\delta^T(q,t),\ \tau_q = t'$
SY' ye bağlı çıkış fonksiyonu		$\delta^C(q,k),\ c_q(j) = n$
Çıkışı olmayan durum		$\delta(q) = \varepsilon,\ \tau_q = 0,\ \mathbf{c}_q = \mathbf{0}$

Örnek 4.1 Bu örnekte $G_{ZS} = (Q, \Sigma, f, T, C, \Delta, q_0)$ şeklinde tanımlanmış olan bir ZS-otomata karşılık gelen durum diyagram gösterimi oluşturulacaktır. G_{ZS}'nin elemanları aşağıdaki gibi tanımlanmış olsun:

Durum ve olay kümeleri:

$$Q = \{q_1, q_2, ..., q_5\},\ \Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma^T, \sigma_1^C, \sigma_2^C, \sigma_3^C, \sigma_4^C\}$$

Durum geçiş fonksiyonu:

$$f(q_1, \sigma_1) = q_2, \quad f(q_2, \sigma_1) = q_5, \quad f(q_2, \sigma_2) = q_2, \quad f(q_2, \sigma_3) = q_2, \quad f(q_2, \sigma^T) = q_1$$

$$f(q_2, \sigma_2^C) = q_4, \quad f(q_2, \sigma_3^C) = q_3, \quad f(q_3, \sigma^T) = q_5, \quad f(q_4, \sigma_4) = q_5, \quad f(q_5, \sigma_1) = q_2$$

Zamanlama Yapısı:

$$T = \{\tau_{q_1}, \tau_{q_2}, ..., \tau_{q_5}\} \text{ ve } \tau_{q_1} = \tau_{q_4} = \tau_{q_5} = 0;\ \tau_{q_2} = 6.6,\ \tau_{q_3} = 2.3$$

Sayma Yapısı:

$$C = \{\mathbf{c}_{q_1}, \mathbf{c}_{q_2}, ..., \mathbf{c}_{q_5}\} \text{ ve } \mathbf{c}_{q_1} = \mathbf{c}_{q_3} = \mathbf{c}_{q_4} = \mathbf{c}_{q_5} = [0 \quad 0 \quad 0 \quad 0];\ \mathbf{c}_{q_2} = [0 \quad 4 \quad 2 \quad 0]$$

Çıkış fonksiyonları:

$$\Delta = \{\delta, \delta^T, \delta^C\}$$

$$\delta(q) = \begin{cases} \varepsilon & q \neq q_4 \\ \sigma_4 & q = q_4 \end{cases}$$

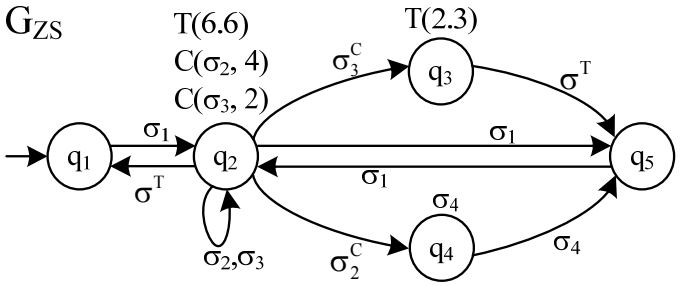
$$\delta^T(q,t) = \begin{cases} \varepsilon & q_2 \text{ ya da } q_3 \text{ aktif degil} \\ \varepsilon & q_i \text{ durumu aktif ve } t \neq t_{q_i} + \tau_{q_i}\ (i = 2,3) \\ \sigma^T \in \Sigma_T & q_i \text{ durumu aktif ve } t = t_{q_i} + \tau_{q_i}\ (i = 2,3) \end{cases}$$

$$\delta^C(q,k) = \begin{cases} \varepsilon & q_2 \text{ durumu aktif degil} \\ \varepsilon & q_2 \text{ durumu aktif ve } AG(\mathbf{c}_{q_2} - \mathbf{n}_{q_2}^k) = AG(\mathbf{c}_{q_2}) \\ \sigma_j^C \in \Sigma_C & q_2 \text{ durumu aktif ve } AG_{\downarrow}(\mathbf{c}_{q_2} - \mathbf{n}_{q_2}^k) = j,\ (j = 2,3) \end{cases}$$

Başlangıç durumu:

$$q_0 = q_1$$

Kapalı şekilde verilen bu ZS-otomatın durum diyagramı gösterimini Çizelge 4.1 yardımıyla Şekil 4.5'teki gibi oluşturmak mümkündür.



Şekil 4.5 : Örnek 4.1'de tanımlanan ZS-otomat

Şekil 4.5'teki ZS-otomatın örnek bir zaman diyagramı Şekil 4.6'da verilmiştir. Bu zaman diyagramına göre başlangıçta q_1 durumunda olan ZS-otomat σ_1 olayı ile q_2 durumuna geçer. q_2 durumu için T zaman yapısı ile 6.6 saniyelik bir gecikme süresi, C sayma yapısı ile σ_2 olayının 4, σ_3 olayının 2 kez sayılacağı tanımlanmıştır. Zaman diyagramından da görüleceği üzere q_2 durumundayken σ_2 ve σ_3 olaylarının olması sayma adımı k'nın değerini bir artırmakta ve (4.13) ifadesi doğrultusunda q_2 durumuna ilişkin $\mathbf{n}_{q_2}$ sayma vektörünün içeriğinin güncellenmesine neden olmaktadır. Ancak σ_2 ve σ_3 için tanımlanan sayma değerlerine ulaşılmadığından herhangi bir sayma olayı üretilmemektedir. q_2 durumu aktif olduktan 6.6 saniye sonra duruma ve ZY'ye bağlı çıkış fonksiyonu tarafından (4.11) ifadesi gereği σ^T zamanlama olayı üretilir ve otomat tekrar q_1 durumuna geçer. q_1 durumundayken σ_1 olayı ile tekrar q_2 durumuna geçilir. q_2 durumundayken t=10.6 anında σ_3 olayının olması sayma adımı k'nın 1 değerini almasına neden olur. Bu durumda t=12.2 anında ikinci kez σ_3 olayının meydana gelmesi ile k=2 değerini alır ve q_2 durumuna ilişkin sayma vektörünün (k=2 için $\mathbf{n}_{q_2}^2$) içeriği anlık olarak [0 0 2 0] olur. Bu ise ($\mathbf{c_2} - \mathbf{n}_{q_2}^2$) vektörünün ağırlık kaybetmesine ve (4.12) ifadesi gereği σ_3^C olayının üretilmesine neden olur. σ_3^C olayı ile otomat q_3 durumuna geçer ve k sayma adımı sıfırlanır.

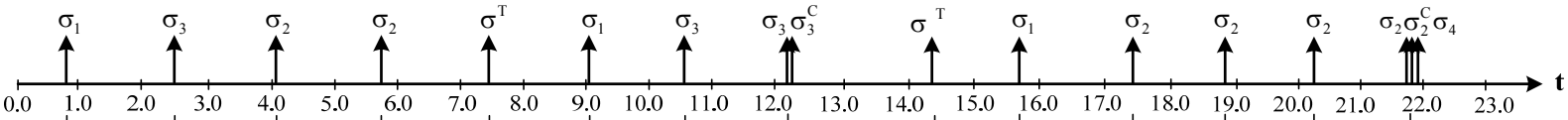
Görüldüğü gibi σ_3 ve σ_3^C olaylarının olması birbirine neden sonuç ilişkisi ile bağlıdır. Önce σ_3 meydana gelmiş, bu ise SY gereği σ_3^C olayının üretilmesine neden olmuştur. İlerleyen bölümlerde tanımlanacak PLC ile gerçekleme yönteminin uygulanmasında görüleceği üzere bu iki olayın oluş anları arasında bir PLC tarama çevrimine karşılık gelen süre söz konudur.

Otomat q_3 durumuna geçtikten 2.3 saniye sonra duruma ve ZY'ye bağlı çıkış fonksiyonu tarafından σ^T olayı üretilir. Bu olayla otomat q_5 durumuna geçer. Bu durumdayken σ_1 olayının olması G_{ZS}'nin tekrar q_2 durumuna geçmesine neden olur. q_2 durumuna ZS yapısı σ_2 olayının 4 kez sayılacağını tanımlamaktadır. Şekil 4.6'daki zaman diyagramından, σ_2 olayı 4 kez sayıldıktan sonra q_4 durumuna geçtiği davranış takip edilebilir.

$Q = \{q_1, q_2, ..., q_5\}$

$\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma^T, \sigma_1^C, \sigma_2^C, \sigma_3^C, \sigma_4^C\}$

$\mathbf{c}_{q_2} = [0 \quad 4 \quad 2 \quad 0]$ $\tau_{q_2} = 6.6$ $\tau_{q_3} = 2.3$



Aktif Durum, q_i	q_1	q_2	q_2	q_2	q_2	q_1	q_2	q_2	q_3	q_5	q_2	q_2	q_2	q_2	q_5
Sayma adımı, k	0	0	1	2	3	0	0	1	2	0	0	0	1	2	3	4	0
Sayma vektörü, n_2^k (q_2 durumu için)	[0 0 0 0]	[0 0 0 0]	[0 0 1 0]	[0 1 1 0]	[0 2 1 0]	[0 0 0 0]	[0 0 0 0]	[0 0 1 0]	[0 0 0 0]	[0 0 0 0]	[0 0 0 0]	[0 1 0 0]	[0 2 0 0]	[0 3 0 0]	[0 0 0 0]
$(\mathbf{c}_2 - \mathbf{n}_2^k)$	[0 4 2 0]	[0 4 2 0]	[0 4 1 0]	[0 3 1 0]	[0 2 1 0]	[0 4 2 0]	[0 4 2 0]	[0 4 1 0]	[0 4 2 0]	[0 4 2 0]	[0 4 2 0]	[0 3 2 0]	[0 2 2 0]	[0 1 2 0]	[0 4 2 0]
$AG(\mathbf{c}_2 - \mathbf{n}_2^k)$	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	1	2

τ_2=6.6 s τ_3=2.3 s

Şekil 4.6 : Örnek 4.1'de tanımlanan ZS-otomat için örnek bir zaman diyagramı

q_4 durumunda duruma bağlı çıkış fonksiyonu tarafından σ_4 olayı üretilmektedir. Bu olay q_4'ün aktif olay kümesi içindedir, dolayısıyla σ_4'ün üretilmesi G_{ZS}'nin q_5 durumuna geçmesine neden olur.

Bu örnekten görüldüğü gibi ayrık olay sistemlerine ilişkin bir tasarımdaki zamanlama ve sayma olayları, gerçekleme aşamasında ayrı bir mekanizmanın kullanılmasına gereksinim olmayacak şekilde ZS-otomat ile ifade edilebilmektedir. Bir sonraki bölümde ZS-otomat ile ifade edilen bir tasarımın gerçeklenmesine yönelik bir yöntem tanıtılmaktadır.

5. GERÇEKLEME

Tasarımın amacı, hedeflenen kontrol davranışını sağlayacak kontrolörü ya da üstdenetleyiciyi belirlemek ve bunu uygun bir gösterim kullanarak ifade etmektir. Önceki bölümlerde ele alınan standart otomat ve bir önceki bölümde tanımı yapılan ZS-otomat, durum diyagramlarını temel alan gösterim şekilleridir. Tasarımın bir gösterim kuralı ile ifadesinden sonraki aşama bir teknolojik araç kullanarak tasarımı gerçeklemektir. Günümüzde kullanılan teknolojik araçlar çoğunlukla mikrokontrolör tabanlı sistemler, birçok durumda da Bölüm 3'te bahsedilen Programlanabilir Lojik Kontrolörler (PLC) olmaktadır. PLC'ler endüstriyel kontrol sistemlerinin otomasyonunda yaygın olarak kullanılmakta, seri üretimin bulunduğu neredeyse bütün tesislerde farklı ölçeklerde uygulama alanı bulmaktadır. Bu nedenle, bir gösterilim şekli ile ifade edilen kontrol tasarımının gerçekleştirileceği fiziksel ortamın birçok durumda PLC'ler olacağı söylenebilir.

Gerçekleme aşamasının sistematik olarak ifade edilmesiyle, ele alınan tasarım probleminden bağımsız bir gerçekleme yöntemi tanımlanmış olur. Böyle bir yöntem gerçeklemenin otomatik olarak yapılması için de kullanılabilir. Burada gerçeklemenin otomatik yapılmasından kasıt, tasarımın bir otomat gösterimini giriş olarak alıp, gerekli PLC kodunu otomatik olarak üreten bir bilgisayar programının kullanılmasıdır. Bu bölümde tanımlanacak olan gerçekleme yönteminden yararlanarak otomatik PLC kod üretimini sağlayan bir uygulama, 5. Bölümde tanıtılacaktır.

Gerek standart otomat tanımında gerekse bir önceki bölümde tanıtılan ZS-otomat tanımında durumlar, olaylar, çıkış ya da durum geçiş fonksiyonları gibi otomat bileşenleri matematiksel kümelerden oluşmaktadır. Ayrıca bir AOS sistemi olarak standart otomat ya da ZS-otomat gösterimleri olayların anlık olması, aynı anda sadece bir olayın olması gibi varsayımları da içermektedir. PLC'ler ise mantıksal işlemleri çevrimsel olarak yerine getiren mikrokontrolör tabanlı sistemlerdir. Bu nedenle, otomat gösterilimi ile ifade edilen bir davranış için, gösterimin bileşenlerini

ve AOS varsayımlarını kabul edilebilir bir yaklaşıkla PLC'de gerçekleyecek bir yöntemin tanımlanmasına gereksinim vardır.

Bu bölümde, Bölüm 4'te tanımlanan ZS-otomatın PLC'lerde gerçeklenebilmesi için bir yöntem tanıtılacaktır. Öncelikle literatürde standart otomat için kullanılmış olan gerçekleme yöntemleri tanıtılacak, AOS ve otomat varsayımlarının PLC'de gerçekleştirilmesinde oluşan problemlerden bahsedilecektir. Bu problemlerden "çığ etkisi" olarak adlandırılan ve literatürde genel bir çözüm geliştirilemeyen sorun için kesin ve sistematik bir çözüm gerçekleme yöntemi ile beraber tanıtılacaktır.

5.1 Standart Gerçekleme Yöntemi

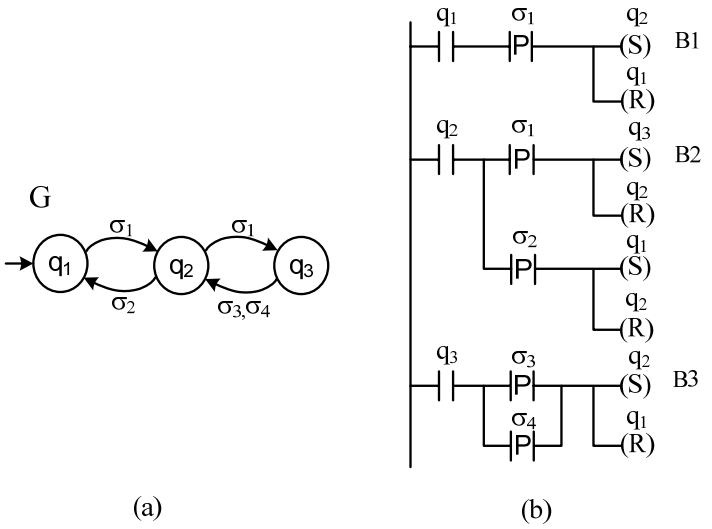
Yukarıda da belirtildiği gibi otomat ya da ZS-otomat matematiksel dünyada durum, olay ve fonksiyon kümeleri ile tanımlanmış yapılar olmasına karşın, Programlanabilir Lojik Kontrolörler mikrokontrolör tabanlı teknolojik araçlardır ve çalışması bellek alanlarında saklanan sayısal verilerin çevrimsel olarak işlenmesi esasına dayanır. Öncelikle bu teknolojik yapının, matematiksel dünyada tanımlanan otomat davranışına en yakın davranışı göstermesi için temel kurallara gereksinim duyulur. "Standart gerçekleme yöntemi" adı ile anılacak olan bu kurallar aşağıda özetlenmiştir. Bu bölümde geliştirilecek olan gerçekleme yöntemi de bu kuralları temel almaktadır.

Durumlar ve olaylar bellek bitleri ile temsil edilebilir. Belirli bir duruma karşılık gelen bir bellek bitinin mantıksal 1 seviyesinde olması otomatın o durumda olduğu anlamına gelecektir. Ayrık olay sistemleri kuramında olayların anlık olduğu varsayımı yapılır. Bu nedenle PLC'lerde bunu karşılamak üzere olaylar işaret seviyelerinin çıkan ya da düşen kenarları ile temsil edilebilir. PLC'lerdeki çıkan ve düşen kenar fonksiyonlarında her tarama çevriminde ilgili işaretin bir önceki değeri ile o anki değeri karşılaştırılır. Eğer mantıksal 0'dan 1'e geçiş belirlenmişse yükselen kenar fonksiyonu, 1'den 0'a geçiş belirlenmişse düşen kenar fonksiyonu tek bir PLC tarama çevrimi için çıkış verecektir. Durum geçiş fonksiyonları içinse bilinen en basit yöntem mantıksal VE ile kurma (SET) / silme (RESET) komutlarının kullanılmasını gerektirmektedir. Bu yöntemde belirli bir duruma karşılık gelen bellek biti ile o durumdan başka bir duruma geçişe neden olaya karşılık gelen bite mantıksal VE işlemi uygulanır ve sonuca göre yeni durum biti kurulur, çıkılan eski durum biti

silinir (sıfırlanır). Şekil 5.1'de örnek bir otomat ve buna karşılık gelen merdiven mantığı ile (Ladder Logic) yazılmış PLC programı verilmiştir. Şekil 5.1b' deki PLC programında q_1, q_2, q_3 otomatın durumlarını temsil eden belek bitleri, σ_1, σ_2, σ_3 ve σ_4 ise çıkan kenarları olaylara karşılık gelen işaretlerdir.

5.1.1 Çığ etkisi sorunu

Yukarıda tanıtılan yöntem kolay uygulanabilir olmasına rağmen otomatın yapısına bağlı olarak davranışı beklenenden farklı olan bir gerçeklemeye neden olabilir. Bu yanlış davranışın nedeni gerçeklenen otomattaki birtakım istenmeyen durum geçişleriyle ilgilidir ve çığ etkisi (avalanche effect) olarak adlandırılır [22]. Esasen, Şekil 5.1a'da görünen otomata karşılık gelen PLC gerçeklemesinde çığ etkisi sorunu vardır. Bu sorunu açıklamak üzere Şekil 5.1b'deki PLC programının çalışması aşağıda detaylı olarak irdelenmiştir.

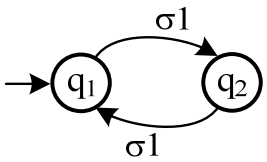


Şekil 5.1 : (a) Örnek bir otomat, (b) bu otomata PLC karşılık gelen PLC programı

G otomatı q_1 durumundayken bu otomata karşılık gelen PLC programında q_1 değişkeni mantıksal 1 seviyesinde olacaktır. Bu durumdayken σ_1 olayı olduğunda kuramsal otomatın q_2 durumuna geçmesi ve ikinci bir σ_1 olayına kadar bu durumda kalması beklenir. Bunun PLC programındaki karşılığı ise q_2 bitinin kurulması q_1 bitinin silinmesidir. Gerçekten de Şekil 5.1b'deki PLC programının B1 ile gösterilen birinci basamağında σ_1 işaretinin yükselen kenarı algılandığında SET ve RESET komutları kullanılarak q_2 biti kurulmakta, q_1 biti ise silinmektedir. Ancak q_2 biti mantıksal 1 seviyesine kurulduğundan ve çıkan kenar fonksiyonu tarama çevrimi boyunca mantıksal 1 seviyesinde kaldığından PLC programının B2 ile gösterilen

ikinci basamağındaki q_3'ü kuran, q_2'yi sıfırlayan SET ve RESET komutları da işletilir. Bu nedenle kuramsal otomatın q_1 durumunda kalması gerekirken bu otomata karşılık gelen PLC programı q_3 durumuna geçmiş olur. Sonuç olarak Şekil 5.1b'de verilen PLC programı Şekil 5.1a'daki otomatı çığ etkisi sorunu nedeniyle hatalı olarak gerçeklemektedir [33].

Çığ etkisi yukarıda ele alınan gerçekleme yöntemi kullanıldığında ardışıl olarak oluşan bir olayın farklı durumlara geçişlere neden olduğu her otomatta karşılaşılabilecek bir sorundur. Bu sorunun çözümü için literatürde PLC programındaki basamak sırasını ters çevirmeye dayalı bir çözüm önerilmiştir [22]. Ancak bu çözümün uygulanması ele alınan probleme göre farklılaşabilmekte, sistematik olmaması nedeniyle çoğu durumda programcının deneyimine bağlı sonuçlar vermekte ve bazı durumlarda ise tamamen çözümü imkânsız kılmaktadır. Şekil 5.2'de σ_1 olay çiftlerini belirlemek için kullanılabilecek basit bir otomat görülmektedir. σ_1 olayının ardışıl olarak durum geçişlerine neden olmasından dolayı çığ etkisi sorunu olan bu otomat için yukarıda ele alınan yöntemde PLC programındaki basamak sırasının değiştirilmesi sorunu çözememektedir.



Şekil 5.2 : Çığ etkisi problemi bilinen yöntemlerle çözülemeyen basit bir otomat

5.2 ZS-otomat İçin PLC ile Gerçekleme Yöntemi

ZS-otomat, standart otomatın durum diyagramı gösterimine ek olarak zamanlama ve sayma özelliklerini de yapısında bulundurmakta, Zamanlama-Sayma Yapısı ve çıkış fonksiyonları da durumlara bağlı olarak ifade edilmektedir. Aşağıda ZS-otomat için tanımlanacak yöntem durum geçişleri ve ilk durumun gerçeklenmesine ek olarak, Zamanlama-Sayma Yapısı ve çıkış fonksiyonlarının gerçeklenmesini de kapsayacaktır. Durum geçişleri ve ilk durumun gerçeklenmesi için verilecek tanımlar kolaylıkla standart otomat için uyarlanabilir. ZS-otomatın standart otomattan ayrılan özelliklerinin gerçeklenmesi ise, PLC'lerin sağladığı hazır kontrol blokları (Zamanlayıcılar ve Sayıcılar) tarafından kullanılabilecek şekilde tanımlanacaktır. Bu

doğrultuda gerçekleme yöntemi aşağıda durum geçişlerinin ve ZS-Yapısının gerçeklenmesine ilişkin olarak iki aşamada verilecektir.

5.2.1 Durum geçişlerinin ve ilk durumun gerçeklenmesi

Daha önce bahsedildiği gibi standart gerçekleme yönteminde çığ etkisi problemini ortadan kaldıran sistematik bir çözümün verilmesi mümkün değildir. Bunun yanında bu yöntem bazı PLC modellerinde normal bit işlem komutlarına göre program belleğinde önemli miktarda daha fazla yer kaplayan kurma ve silme (SET ve RESET) komutlarını yoğun olarak kullanmaktadır. Örneğin Siemens S7-200 PLC'lerde standart bir bit işlem komutu için program belleğinde 2 byte'lık bir alan ayrılırken, SET ve RESET komutlarının her biri için 7 byte'lık alan ayrılır. Ayrıca kurma ve silme komutlarının kullanılması herhangi bir duruma ilişkin geçiş fonksiyonlarını PLC programından takibini de güçleştirmektedir. Örneğin Şekil 5.1b'de q_2 durumuna ilişkin toplam dört adet SET veya RESET komutu kullanılmıştır. Bu ise q_2 durumuna geçiş fonksiyonlarını belirleyebilmek için bu dört komuta karşılık gelen koşulların incelenip elde edilen bilginin birleştirilmesini gerektirir. Özellikle hata bulma ya da programda yapılacak olan değişikliklerde bu özellik zorluklara neden olacaktır.

Aşağıda matematiksel olarak tanıtılacak, ilerleyen paragraflarda ise pratik uygulanabilirliği irdelenecek yöntem, temel olarak kurma ve silme komutlarının kullanılmadığı, çığ etkisi problemi için kesin bir çözüm oluşturan ve takip edilmesi, değişikliklerin uygulanması kolay olan bir yapı oluşturmaktadır. İlk aşamada kurma ve silme komutları bit işlem komutlarıyla gerçeklenecektir.

Gerçekleme, PLC'nin çalışma prensibine uygun olarak zamanda ayrık anlarda işletilen mantıksal fonksiyonlarla ifade edilecektir. Bir mantıksal q değişkeninin kurma koşulu S ile, silme koşulu R ile temsil edilsin. $k \in \mathbf{N}$ olmak üzere q değişkeninin mevcut değeri q(k), bir sonraki değeri q(k+1) ile gösterilirse

$$q(k+1) = S + q(k).\overline{R} \qquad \textbf{(5.1)}$$

mantıksal fonksiyonunun q'yu S ile kurduğu, R ile sıfırladığı kolaylıkla gösterilebilir. Burada '+' mantıksal VEYA işlemine, "." VE işlemine, $\overline{R}$ ise R'nin mantıksal DEĞİL'ine karşılık gelmektedir. Bu mantıksal fonksiyon verilen bir otomatın durum geçiş koşullarının gerçeklenmesi için de kullanılabilir. Aşağıdaki

tanımlar ZS-otomatın durum geçiş koşullarının mantıksal fonksiyonlarla ifade edilebilmesine yönelik olarak verilmiştir. Tanımlar, ZS-otomat için verilmiş olmasına rağmen Zamanlama ve Sayma yapısı ile ilgili olmayan tanımlar kolaylıkla standart otomat için uyarlanabilir [33].

Tanım 5.1. $G_{ZS} = (Q, \Sigma, f, \Gamma, T, C, \Delta, q_0)$ ZS-otomatı verilsin. $Q = \{q_1, q_2, ..., q_{N_Q}\}$ durum kümesindeki her duruma 1 ile N_Q arasında bir tam sayı karşılık gelecek şekilde oluşturulan $I_Q = \{1, 2, ..., N_Q\}$ kümesine G_{ZS}'ye ilişkin *durum indis kümesi* denir.

Tanım 5.2. $G_{ZS} = (Q, \Sigma, f, \Gamma, T, C, \Delta, q_0)$ ZS-otomatı verilsin. $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_{N_\Sigma}\}$ durum kümesindeki her duruma 1 ile N_Σ arasında bir tam sayı karşılık gelecek şekilde oluşturulan $I_\Sigma = \{1, 2, ..., N_\Sigma\}$ kümesine G_{ZS}'ye ilişkin *olay indis kümesi* denir.

Tanım 5.3. Durum geçiş fonksiyonu f olan bir ZS-otomata ilişkin durum indis kümesi $I_Q = \{1, 2, ..., N_Q\}$, olay indis kümesi $I_\Sigma = \{1, 2, ..., N_\Sigma\}$ olsun. q_i *durumuna geçişe ilişkin durum indis kümesi* şu şekilde tanımlanır:

$$I_{SQ}(i) = \{j \in I_Q \mid \exists k \in I_\Sigma, f(q_j, \sigma_k) = q_i\}, \forall i \in I_Q \qquad \textbf{(5.2)}$$

Bu tanıma göre, $I_{SQ}(i)$ kümesinin elemanları q_i durumuna geçişin mümkün olduğu durum indislerine karşılık gelir.

Tanım 5.4. Durum geçiş fonksiyonu f olan bir ZS-otomata ilişkin durum indis kümesi $I_Q = \{1, 2, ..., N_Q\}$, olay indis kümesi $I_\Sigma = \{1, 2, ..., N_\Sigma\}$ olsun. q_i *durumuna geçişe neden olan olay indis kümesi* şu şekilde tanımlanır:

$$I_{S\Sigma}(i) = \{j \in I_\Sigma \mid \exists k \in I_Q, f(\sigma_k, \sigma_j) = q_i\}, \forall i \in I_q \qquad \textbf{(5.3)}$$

Tanım 5.3'e benzer şekilde $I_{S\Sigma}(i)$ kümesinin elemanları q_i durumuna geçişi sağlayan tüm olay indislerine karşılık gelir.

Tanım 5.5. Durum geçiş fonksiyonu f, aktif olay fonksiyonu Γ olan bir ZS-otomata ilişkin durum indis kümesi $I_Q = \{1, 2, ..., N_Q\}$, olay indis kümesi $I_\Sigma = \{1, 2, ..., N_\Sigma\}$ olsun. q_j *durumundan* q_i *duruma geçişe neden olan olay indis kümesi*

$$I_{T\Sigma}(i, j) = \{k \in \Sigma \mid k \in \Gamma(q_j) \wedge k \in I_{S\Sigma}(i)\}, \quad \forall (i, j) \in I_{SQ} \times I_{SQ} \qquad \textbf{(5.4)}$$

ile verilir.

Tanım 5.6. Durum geçiş fonksiyonu f, aktif olay fonksiyonu Γ olan bir ZS-otomata ilişkin durum indis kümesi $I_Q = \{1, 2, ..., N_Q\}$, olay indis kümesi $I_\Sigma = \{1, 2, ..., N_\Sigma\}$ olsun. q_i *durumundan çıkışa neden olan olay indis kümesi*

$$I_{R\Sigma}(i) = \{j \in I_\Sigma \mid \sigma_j \in \Gamma(q_i)\}, \ \forall i \in I_Q \qquad \textbf{(5.5)}$$

şeklinde tanımlanır.

Verilen tanımların bir uygulaması olarak aşağıdaki örnek göz önüne alınacaktır.

Örnek 5.1. Şekil 5.1a' da verilen otomat için durum indis kümesi $I_Q = \{1,2,3\}$, olay indis kümesi $I_\Sigma = \{1,2,3,4\}$' tür. Tanım 5.3, 5.4, 5.5 ve 5.6, q_2 durumu için uygulanırsa aşağıdaki kümeler elde edilir.

q_2 *durumuna geçişe ilişkin durum indis kümesi* $I_{SQ}(2) = \{1,3\}$; q_2 *durumuna geçişe neden olan olay indis kümesi* $I_{S\Sigma}(2) = \{1,3,4\}$, $q_j \ (\forall j \in I_Q)$ *durumlarından* q_2 *durumuna geçişe neden olan olay indis kümeleri*; $I_{T\Sigma}(2,1) = \{1\}$, $I_{T\Sigma}(2,2) = \phi$, $I_{T\Sigma}(2,3) = \{3,4\}$; q_2 *durumundan çıkışa neden olan olay indis kümesi* $I_{R\Sigma}(2) = \{1,2\}$ ■

Daha önce de belirtildiği gibi durum geçiş fonksiyonlarının gerçeklenmesi için (5.1) ilişkisi ile verilen mantıksal fonksiyondan faydalanılması amaçlanmaktadır. Bu nedenle durum geçiş koşullarının Eşitlik 5.1'de görülen Set ve Reset koşullarına karşılık getirecek tanımlara gereksinim vardır. Bu tanımlar aşağıda mantıksal fonksiyonlar kullanılarak verilmektedir. Tanımlarda kullanılacak mantıksal değişkenlere ilişkin semboller, ZS-otomatta durumlar ve olaylara ilişkin kullanılan sembollerle aynı olacaktır. Bu değişkenlerin bir k adımındaki değerleri ise $q_i(k)$ ve $\sigma_i(k)$ ile gösterilmektedir. "+" sembolü (ya da toplam operatörü $\sum$) mantıksal VEYA, "." sembolü (ya da çarpım operatörü $\prod$) mantıksal VE işlemine karşılık gelecektir.

Tanım 5.7. q_i *durumuna ilişkin kurma koşulunun k adımındaki mantıksal ifadesi*

$$S_i(k) = \sum_{j \in I_{SQ}(i)} q_j(k) . \sigma(i,j) \qquad \textbf{(5.6)}$$

ile verilir. Burada

$$\sigma(i, j) = \sum_{n \in I_{T\Sigma}(i,j)} \sigma_n \qquad \textbf{(5.7)}$$

şeklinde tanımlanır ve q_j durumundan q_i durumuna geçişe neden olan tüm olayların eşdeğerine karşılık gelir.

Tanım 5.8. q_i *duruma ilişkin silme koşulunun mantıksal ifadesi*

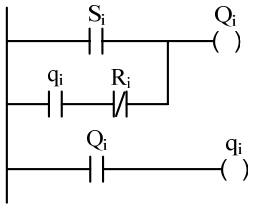
$$R_i(k) = \sum_{j \in I_{R\Sigma}(i)} \sigma_j(k) \qquad \textbf{(5.8)}$$

ile verilir.

Tanım 5.7 ve 5.8 ile *i* indisli duruma ilişkin kurma ve silme koşulları tanımlandığından bu duruma ilişkin mantıksal ifade Eşitlik 5.1'den faydalanılarak şu şekilde elde edilebilir:

$$q_i(k+1) = S_i(k) + q_i(k).\overline{R_i(k)} \qquad \textbf{(5.9)}$$

Eşitlik 5.9'a ilişkin PLC programı kolaylıkla merdiven dili programlama tekniği ile elde edilebilir. $q_i(k+1)$ mantıksal değişkeni Q_i, $q_i(k)$ mantıksal değişkeni q_i ile gösterilirse Eşitlik 5.9'a karşılık gelen PLC programı Şekil 5.3'teki gibi verilebilir. Programın son basamağında Q_i değişkeninin içeriği q_i değişkenine atanmaktadır. Buna göre PLC programındaki q_i mantıksal değişkeni, q_i durumunun mevcut PLC tarama çevrimindeki; Q_i mantıksal değişkeni ise bu durumun bir sonraki PLC çevrimindeki gösterimine karşı geldiği yorumu yapılabilir.



Şekil 5.3 : Eşitlik 5.9'a karşılık gelen merdiven dili programı.

Örnek 5.2. Örnek 5.1'deki otomat tekrar ele alınsın. Tanım 5.5 ve Tanım 5.6'ya göre bu otomatın q_2 durumu için

$$S_2(k) = \sum_{j \in I_{SQ}(2)} q_j(k).\sigma(2,j) = q_1(k).\sigma_1 + q_3(k).(\sigma_3 + \sigma_4) \qquad \textbf{(5.10)}$$

$$R_2(k) = \sum_{j \in I_{R\Sigma}(2)} \sigma_j(k) = \sigma_1 + \sigma_2 \qquad \textbf{(5.11)}$$

ifadeleri elde edilir. Bu ifadeler Eşitlik 5.9'da yerine konulursa q_2 durumuna ilişkin mantıksal ifade

$$q_2(k+1) = q_1(k).\sigma_1 + q_3(k).(\sigma_3 + \sigma_4) + q_2(k).\overline{\sigma_1}.\overline{\sigma_2} \quad \textbf{(5.12)}$$

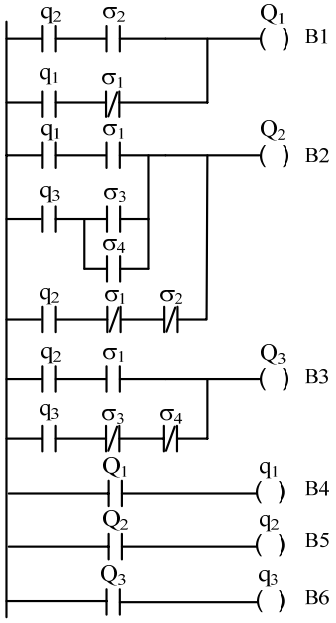
şeklinde elde edilir. Benzer şekilde q_1 ve q_3 durumlarına ilişkin mantıksal ifadeler

$$q_1(k+1) = q_2(k).\sigma_2 + q_1(k).\overline{\sigma_1} \quad \textbf{(5.13)}$$

ve

$$q_3(k+1) = q_2(k).\sigma_1 + q_3(k).\overline{\sigma_3}.\overline{\sigma_4} \quad \textbf{(5.14)}$$

olarak elde edilir. Eşitlik 5.12, 5.13 ve 5.14 kullanılarak, ele alınan otomata ilişkin PLC programı Şekil 5.4'teki gibi geliştirilebilir. Programda olaylara karşılık gelen bellek bitlerinin ilgili işaretlerin yükselen ya da düşen kenarları ile elde edilmiş olduğu varsayılmaktadır.



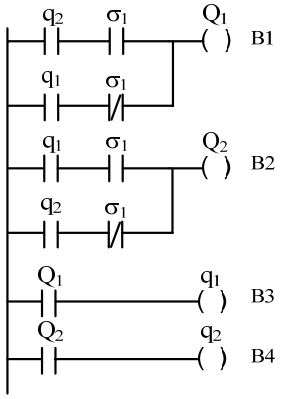
Şekil 5.4 : Örnek 5.2' ye ilişkin PLC programı

Hatırlanacağı üzere Örnek 5.1 ve Örnek 5.2'de ele alınan otomat Bölüm 5.1.1'de bahsedilen programlama yöntemi ile gerçeklendiğinde çığ etkisi sorunu ile karşılaşılmaktaydı. Bu bölümde tanıtılan gerçekleme yöntemi ile çığ etkisi sorunu çözülmektedir: Şekil 5.4'te verilen PLC programı incelendiğinde q_1 durumundan q_2 durumuna geçişin gerçeklendiği B2 basamağında σ_1 olayının olmasıyla Q_2 bellek bitinin mantıksal 1 seviyesine kurulduğu görülür. B3 basamağında ise Q_3 bellek bitinin kurulması q_2 ve σ_1 değişkenlerinin mantıksal 1 seviyesinde olmasına bağlıdır. q_2 değişkeni için ayrı bir bellek biti kullanılması nedeniyle B3 basamağı,

değeri B2'de belirlenen Q_2 değişkeninden etkilenmez ve Q_3 bitinin mantıksal 1 seviyesine kurulması, yani çığ etkisi engellenmiş olur.

Ele alınan örnek için yukarıda yapılan irdeleme, verilen bir otomattaki durumlara ilişkin mevcut PLC tarama çevrimi ve bir sonraki tarama çevrimi için farklı bit değişkenlerinin kullanılması sayesinde çığ etkisi sorununun giderildiğini göstermektedir. Açıktır ki kontrol kurallarını ifade eden otomatın programlanmasında her durum için iki farklı bit değişkenin kullanılması PLC'nin bellek kullanımında bir miktar artışa neden olacaktır. Durum sayısı çok büyük olmayan kontrolör ya da üstdenetleyicilerin gerçeklenmesinde genel olarak bellek kullanımındaki bu artış kabul edilebilir sınırlar içinde olmaktadır. Esasen bir çok uygulamada gerek sistem modelinin, gerekse kontrolör modelinin tek bir otomatla ifadesi, modelin ya da tasarımın çok sayıda durum içermesine neden olmaktadır. Bu nedenle, çoğunlukla kontrolör modelini toplamda daha az durumla ifade etmek üzere, Bölüm 3'te bahsedilen modüler kontrol, lokal modüler kontrol gibi yapılar kullanılmaktadır. Bu yapılarla, kontrolör modelindeki toplam durum sayısı önemli oranda azaltılmakta, böylece kontrolör durumlarının gerçeklenmesinde bellek kullanımı problemi büyük ölçüde önemini yitirmektedir [33].

Daha önce de belirtildiği gibi Şekil 5.2' de verilen otomatın standart yöntem ile gerçeklenmesinde çığ etkisi problemi çözülememektedir. Şekil 5.5'te bu bölümde tanımlanan yöntemle gerçeklenen ve çığ etkisi sorunu bulunmayan PLC programı görülmektedir.



Şekil 5.5 : Şekil 5.2' deki otomat için çığ etkisi problemi olmayan gerçekleme

PLC ile gerçeklemede kullanılan yöntemin kolay uygulanabilir olması önemlidir. Bu nedenle yukarıda matematiksel tanımlarla verilen yöntem uygulamaya yönelik olarak da değerlendirilmelidir. Kolay uygulanabilirliği irdelemek üzere Şekil 5.1a'daki

otomatı ve Şekil 5.4'te bu otomata karşılık gelen PLC programı tekrar ele alınsın. Şekil 5.1a'da görüldüğü gibi otomatın q_1 durumuna geçmesi için q_2 durumunda olması *ve* σ_2 olayının meydana gelmesi gerekir. Bunun Şekil 5.4'teki PLC programında karşılığı, B1 basamağında görülen ve q_2 ile σ_2 değişkenleri tarafından oluşturulan "$q_2 \text{VE} \sigma_2$" mantıksal ifadesidir. Bu ise Tanım 5.7 ile verilen kurma koşuluna karşılık gelir. Diğer taraftan, Şekil 5.1a'daki otomatın q_1 durumunda kalması için otomat bu durumdayken σ_1 olayının olmaması gerektiği açıktır. Bunun Şekil 5.4'teki PLC programındaki karşılığı B1 basamağında q_1 ile σ_1 değişkenleri ile oluşturulan "q_1 VE $\overline{\sigma_1}$" mantıksal ifadesidir. Bu ise matematiksel olarak Tanım 5.8 ve Eşitlik 5.9'a karşılık gelmektedir. Benzer yorumlar otomatın q_2 ve q_3 durumlarına karşılık gelen B2 ve B3 basamakları için de yapılabilir. Esasen Tanım 5.7, Tanım 5.8 ve Eşitlik 5.9 ile matematiksel olarak ifade edilen bu yöntem, verilen bir otomatın her bir durumuna ilişkin geçiş koşullarının ve otomatın o durumda kalması için gerekli koşulların gerçeklenmesine dayanmaktadır. Bu açıdan yorumlandığında, matematiksel olarak tanıtılan bu yöntemin kolay uygulanabilir olma özelliğinden bahsedilebilir.

Bölüm 5.2'de bahsedilen yöntemde bir duruma ilişkin geçiş koşulları için farklı basamaklarda, farklı sayılarda kurma ve silme komutları kullanılabilmektedir. Örneğin Şekil 5.1b'de q_2 durumu için 3 ayrı basamakta toplam 4 adet SET ya da RESET komutu kullanılmıştır. Bu ise programı okumayı, hata bulma ya da değişiklik yapma gibi işlemleri oldukça güçleştirmektedir. Bu çalışmada tanıtılan yöntemde ise her bir durumun geçiş koşullarının ifadesi için PLC programında tek bir basamak yeterli olmaktadır. Şekil 5.4'ten de görüleceği üzere otomatın q_1, q_2 ve q_3 durumlarına PLC programında sırasıyla B1, B2 ve B3 basamakları karşı düşmektedir. Bu nedenle kolay uygulanabilirliğin yanında, bu yöntemle elde edilen PLC programının yapısal olarak kolay okunabilir olma özelliğinin olduğu da söylenebilir.

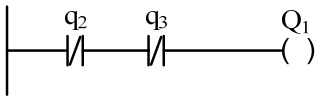
İlk durumun programlanması:

Tam bir gerçekleme için durum geçişlerinin yanında otomatın ilk durumunun da programlanması gereklidir. Bunun için PLC'nin ilk tarama çevriminde mantıksal 1 değeri alan özel belek alanlarından faydalanmak mümkündür. Örneğin S7-200

PLC'sinde SM0.1 ile adreslenen özel bellek biti program işletilmeye başladığı anda ilk duruma karşılık gelen bit değişkenini kurmak için kullanılabilir. Bunun dışında kullanılacak olan PLC'nin böyle bir özelliğinden bağımsız olarak "otomat hiçbir durumda değilse ilk durumdadır" mantığı ile de ilk durumun programlanması mümkündür. Bu sözel ifade, i_0 ilk durumun indis numarası olmak üzere, formal olarak aşağıdaki mantıksal eşitlikle verilebilir.

$$q_0(k+1) = \prod_{i \in I_Q \setminus i_0} \overline{q(k)} \qquad \textbf{(5.15)}$$

Buna göre Şekil 5.1a' da verilen otomat için ilk duruma ilişkin mantıksal fonksiyon $Q_1 = \overline{q_2}.\overline{q_3}$ şeklinde olacaktır. Bu mantıksal fonksiyonu gerçekleyen PLC programı ise Şekil 5.6'da verilmiştir.



Şekil 5.6 : İlk durum için PLC gerçeklemesi

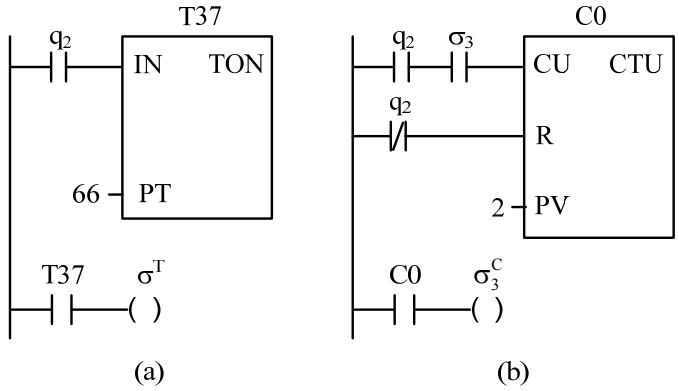
Şekil 5.4'teki B1 basamağı yerine Şekil 5.6'daki ilk duruma karşılık gelen program parçasının kullanılması gerçekleme açısından hiçbir probleme neden olmayacaktır. Çünkü Şekil 5.6'da görülen programa göre q_2 durumundan ya da q_3 durumundan çıkılıp q_1 durumuna geçilen her koşulda Q_1 değişkeni mantıksal 1 seviyesine kurulacak, yani PLC gerçeklemesinde istenilen davranış sağlanmış olacaktır. Bu ise ilk durumun programlanması için PLC'de daha fazla program belleği kullanmak yerine program belleğinden kazanım anlamına gelmektedir.

5.2.2 ZS-yapısının ve çıkış fonksiyonların gerçeklenmesi

Gerçekleme yöntemi için buraya kadar verilen tanımlar durum geçişleri ve ilk durumun programlanabilmesi ile ilgilidir ve standart otomat gösteriminin gerçeklenmesi için de kolaylıkla uyarlanabilir. ZS-otomatı standart otomattan ayıran zamanlama ve sayma yapıları (ZS-yapısı) ile çıkış fonksiyonlarının gerçeklenmesine ilişkin yöntemlerin de tanımlanmasına gereksinim vardır. Bunların gerçeklenmesi için PLC'lerde zamanlama ve sayma işlemlerini yerine getiren hazır program bloklarından, yani Zamanlayıcı ve Sayıcılardan faydalanılacaktır. Bölüm 3'te Zamanlayıcı ve Sayıcı program bloklarının işlevleri ve bu blokların giriş-çıkışları

tanımlanmıştı. ZS-Yapısı ve ilgili çıkış fonksiyonlarının gerçekleme yöntemi de bu blokların tanımındaki giriş çıkışlar ile verilebilir.

Sıfırdan farklı bir gecikme öngörülen bir durum için Şekil 5.7'de verilen Gecikmeli Kapatan Zamanlayıcının (TON) "IN" girişine, bu durumu temsil eden mantıksal değişken bağlanmalıdır. "PT" girişine ise Bölüm 3.3.1'de anlatıldığı şekilde gecikme süresine karşılık gelen bir tamsayı değeri atanmalıdır. Bir q durumu için Zamanlama Yapısında τ_q ile tanımlanan gecikme süresi, "PT" girişine $10 \times \tau_q$ değerini girerek T37-T63, T101-T255 zamanlayıcılarından biri ile saniye cinsinden gerçeklenebilir. Şekil 4.5'te verilen ZS-otomat için Zaman Yapısı ile tanımlanan $\tau_{q_2} = 6.6$ değeri ve buna ilişkin çıkış fonksiyonunun PLC gerçeklemesi Şekil 5.7a'da verilmiştir. Zamanlayıcı çıkışı, zamanlama olayına karşılık gelen σ^T değişkenine atanmaktadır. Zamanlama olayının meydana gelmesi Şekil 4.5'e göre q_1'e geçişle sonuçlanacağından, PLC programında σ^T'nin mantıksal 1 değerini alması bir sonraki PLC çevriminde q_2'nin, dolayısıyla T37 zamanlayıcısının sıfırlanmasına neden olacaktır. Böylece σ^T değişkeni olayların gerçeklenmesine uygun olarak, bir tarama çevrimi boyunca mantıksal 1 değerini almış olacaktır.



Şekil 5.7 : ZS-yapısına ilişkin çıkış fonksiyonlarının gerçeklenmesi

Şekil 3.9'da S7-200 PLC'leri için verilen İleri Sayıcının (CTU) "CU" girişine, sayımı yapılacak olaya ve sayımı yapılması gereken duruma göre atama yapılmalıdır. Sayma Yapısı ile bir q_i durumu için tanımlanan $\mathbf{c}_{q_i}$ vektörü için $c_{q_i}(j) \neq 0$ koşulu σ_j olayının q_i durumda $c_{q_i}(j)$ kez sayılacağına karşılık gelir. Bu koşul CTU sayıcısının "CU" girişine $q_i.\sigma_j$ mantıksal fonksiyonu atanmasıyla gerçeklenebilir. İlgili durumdan çıkılmasıyla da sayıcıyı sıfırlayan "R" girişine mantıksal 1 değeri

atanmalıdır. Bu ise $\overline{q_i}$ ile gerçeklenebilir. Son olarak CTU sayıcısının "PV" girişi için Sayma Yapısı ile tanımlana $c_{q_i}(j)$ değeri atanmalıdır. Şekil 4.5'te verilen ZS-otomat için Zaman Yapısı ile tanımlanan $c_{q_2}(3) = 2$ değerine ilişkin çıkış fonksiyonunun PLC gerçeklemesi Şekil 5.7b'de verilmiştir. Görüldüğü gibi C0 sayıcısının çıkışı σ_3^C sayma olayına atanmaktadır.

6. OTOMATİK PLC KOD ÜRETİMİ

Önceki bölümlerde geliştirilen zamanlama ve sayma davranışlarını içeren durum diyagramı tabanlı gösterim biçimi ve gerçekleme yöntemi, tasarımın fiziksel bir araç ile gerçeklenmesi için sistematik bir altyapı sağlamaktadır. Bu sistematik yöntemler kümesi tasarım ile hedeflenen davranışın doğru bir şekilde gerçeklenmesinin yanı sıra, programlanabilir olma özelliğini de taşımaktadır. Bu bölümde bu özelliğinden faydalanarak önceki bölümlerde tanımlanan yöntemle tasarımı gerçekleyen PLC kod kümesini üreten bir Matlab uygulaması tanıtılacaktır. *OtoPLC* olarak adlandırılan bu uygulamaya ilişkin temel program listeleri aşağıda tanıtılmaktadır. Uygulamanın tamamı için EK B'de verilen otoplc.m dosyası görülmelidir.

6.1 OtoPLC: Otomatik PLC Kod Üretimi İçin Geliştirilen Matlab Uygulaması

OtoPLC uygulaması, ZS-otomat ile ifade edilen tasarımı (4.6) ifadesine uygun bir şekilde giriş olarak alıp, bu tasarıma karşılık gelen PLC kodunu çıkış olarak üretmektedir. Bu amaçla durum geçiş fonksiyonu, zamanlama ve sayma yapısı gibi verilerin Matlab ortamında uygun ifadelerinin oluşturulması gerekmektedir. Aşağıda (4.6) ifadesi ile tanımlanan ZS-otomat parametrelerinin Matlab ortamına uygun matris ya da vektör ifadeleri tanıtılmaktadır.

6.1.1 Durum geçiş fonksiyonu, çıkış fonksiyonu ve ZS-yapısının tanımlanması

Belirli bir kontrol davranışına yönelik tasarım

$$G_{ZS} = (Q, \Sigma, f, \Gamma, T, C, \Delta, q_0) \qquad \textbf{(6.1)}$$

ZS-otomatı ile ifade edilmiş olsun. Bu otomata ilişkin durum geçiş fonksiyonunun taşıdığı bilgi $N_Q \times N_\Sigma$ boyutlu bir matrisle ifade edilebilir. Böyle bir **F** matrisinin **F**(i,j) elemanı, otomat q_i durumundayken σ_j olayının olması halinde geçilecek yeni durumun Tanım 5.1 ile verilen indis numarasına karşılık gelir. (4.14), (4.15) ve (4.16) ifadelerine uygun olarak **F** matrisi boyutları sırasıyla $N_Q \times N_{\Sigma_S}$, $N_Q \times 1$ ve $N_Q \times N_{\Sigma_S}$ olan $\mathbf{F_S}$, $\mathbf{F_T}$ ve $\mathbf{F_C}$ alt matrislerine bölünebilir. $\mathbf{F_S}$ sadece sistem olayları,

$\mathbf{F_T}$ sadece zamanlama olayı, $\mathbf{F_C}$ ise sadece sayma olayları ile ilgili olan durum geçişlerine karşılık gelmektedir. Açıktır ki **F** matrisi ile $\mathbf{F_S}$, $\mathbf{F_T}$ ve $\mathbf{F_C}$ matrisleri arasında (4.17) ilişkisine paralel olarak

$$\mathbf{F} = [\mathbf{F_S} \;\; \mathbf{F_T} \;\; \mathbf{F_C}] \tag{6.2}$$

ilişkisi geçerlidir. Buna göre Şekil 4.5'te verilen örnek ZS-otomat için $\mathbf{F_S}$, $\mathbf{F_T}$ ve $\mathbf{F_C}$ matrisleri aşağıdaki gibi olacaktır.

$$\mathbf{F_S} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 5 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \\ 2 & 0 & 0 & 0 \end{bmatrix}, \; \mathbf{F_T} = \begin{bmatrix} 0 \\ 1 \\ 5 \\ 0 \\ 0 \end{bmatrix}, \; \mathbf{F_C} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \; \mathbf{F} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & 2 & 0 & 1 & 0 & 4 & 3 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.3}$$

Zamanlama ve sayma yapıları da (4.7) ve (4.8) ifadelerine uygun şekilde Matlab ortamında vektör ve matris değişkenleri olarak tanımlanabilir. Şekil 4.5'te verilen örnek ZS-otomat için zamanlama yapısı ve sayma yapısı aşağıdaki **T** vektörü ve **C** matrisi ile ifade edilecektir.

$$\mathbf{T} = \begin{bmatrix} 0 & 6.6 & 2.3 & 0 & 0 \end{bmatrix}, \; \mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.4}$$

Çıkış fonksiyonları kümesinin, (4.9) ilişkisinde verildiği gibi üç bileşeni bulunmaktadır. Bunlardan Zamanlama ve Sayma yapıları ile ilgili olanlar daha önce Bölüm 5.2.2'de belirtildiği gibi PLC'nin zamanlama ve sayma fonksiyonları ile gerçekleştirilecektir. İlerleyen satırlarda Matlab uygulaması ile PLC'nin Zamanlayıcı ve Sayıcı fonksiyonlarının nasıl programlandığı gösterilecektir. Duruma bağlı çıkış fonksiyonu ise yine bir vektör değişkeni ile ifade edilebilir. Buna göre **δ** vektörünün *i*. elemanı, q_i durumunda üretilecek olayın Tanım 5.2 ile verilen indisini göstermek (0, boş kelime ε'na karşılık gelir) üzere Şekil 4.5'teki ZS-otomata ilişkin duruma bağlı çıkış fonksiyonu,

$$\boldsymbol{\delta} = \begin{bmatrix} 0 & 0 & 0 & 4 & 0 \end{bmatrix} \tag{6.5}$$

şeklinde ifade edilecektir.

Ele alınan örnek ZS-otomat için ilk durum indisinin 1 olduğu göz önüne alınırsa buraya kadar belirlenen tüm matris, vektör ve skalar değişkenlerin OtoPLC uygulamasındaki tanımlamaları aşağıdaki gibi olacaktır.

```
%durum gecisleri
Fs(1,:)=[2 0 0 0];
Fs(2,:)=[5 2 2 0];
Fs(3,:)=[0 0 0 0];
Fs(4,:)=[0 0 0 5];
Fs(5,:)=[2 0 0 0];
%zamanlama olayi ile ilgili durum geçişleri
Ft=[0 1 5 0 0]';
%sayma olaylari ile ilgili durum geçişleri
Fc(1,:)=[0 0 0 0];
Fc(2,:)=[0 4 3 0];
Fc(3,:)=[0 0 0 0];
Fc(4,:)=[0 0 0 0];
Fc(5,:)=[0 0 0 0];
%durum geçiş fonksiyonu
F=[Fs Ft Fc];
%zamanlama yapısı
T=[0 6.6 2.3 0 0];
%sayma yapısı
C(1,:)=[0 0 0 0];
C(2,:)=[0 4 2 0];
C(3,:)=[0 0 0 0];
C(4,:)=[0 0 0 0];
C(5,:)=[0 0 0 0];
%duruma bagli cikis fonksiyonu
Delta=[0 0 0 4 0];
%sistem olayları sayısı
Nsigmas=length(Fc(1,:));
%ilk durum
ilk_durum=1;
```

Şekil 6.1 : Matlab uygulamasında örnek ZS-otomata ilişkin tanımlamalar.

6.1.2 Başlangıç değerlerinin atanması ve özçevrimlerin silinmesi

Şekil 6.1 ile tanımlanan verilerle Tanım 5.1 ve 5.2 ile verilen I_Q durum indis kümesi ve I_Σ olay indis kümeleri oluşturulabilir. PLC gerçeklemesinde özçevrimlerin programlanması anlamlı olmayacağından **F** matrisinde özçevrimlere karşılık gelen veriler silinebilir. OtoPLC uygulamasındaki durum ve olay indis kümelerinin oluşturulması, PLC için zamanlayıcı ve sayıcı bloklarına ilişkin ilk değerler atamaları ve özçevrimleri tespit edilip silinmesine karşılık gelen program parçası Şekil 6.2' de verilmiştir.

```
%ilk degerlerin ve indis kümelerinin oluşturulması
Nq=length(F(:,1));
Nsigma=length(F(1,:));
Iq=1:Nq;
Isigma=1:Nsigma;
timer_no=36;
counter_no=0;
%ozcevrim belirleme ve silme
for i=1:Nq
   for j=1:Nsigma
      if F(i,j)==i
         F(i,j)=0;
      end
   end
end
```

Şekil 6.2 : İlk değerlerin atanması ve öz çevrimlerin silinmesi

6.1.3 Geçişlere ilişkin durum ve olay indis kümelerinin atanması

Durum geçiş fonksiyonları için Tanım 5.3, Tanım 5.5 ve Tanım 5.6 ile verilen, I_{SQ}, $I_{T\Sigma}$ ve $I_{R\Sigma}$ indis kümelerinin Matlab ortamında oluşturulması gerekir. İndis numarası *i* olan q_i durumu için Şekil 6.3'te görülen program parçasında Isq(i) vektörünün elemanları, Tanım 5.3 ile verilen $I_{SQ}(i)$ kümesinin elemanlarını oluşturmaktadır. Benzer şekilde Matlab uygulamasında Itsigma(i) vektörünün elemanları $I_{T\Sigma}(i)$ kümesinin, Irsigma(i) vektörünün elemanları da $I_{R\Sigma}(i)$ kümesinin elemanlarını oluşturmaktadır. Şekil 6.3'te bunu sağlayan program parçası görülmektedir.

```
%qi ye geçişe ilişkin durum indis kümesi,
%qj den qi ye geçişe neden olan olay indis kümesi,
Isq=zeros(Nq,1);
Itsigma=zeros(Nq,1);
Irsigma=zeros(Nq,1);
for m=1:Nq
   durum_indis=0;
   for i=1:Nq
      for j=1:Nsigma
         if F(i,j)==m
            durum_indis=durum_indis+1;
            Isq(m,durum_indis)=i;
            Itsigma(m,i)=j;
          end
       end
    end
%Reset koşulları
   olay_indis=0;
   for j=1:Nsigma
      if F(m,j)~=0
         olay_indis=olay_indis+1;
         Irsigma(m,olay_indis)=j;
      end
   end
end
```

Şekil 6.3 : I_{SQ}, $I_{T\Sigma}$ ve $I_{R\Sigma}$ indis kümelerinin oluşturulması

```
%set kosulu ifadesi
for i=1:Nq
      basla=0;
      for j=1:length(Isq(i,:));
         if Isq(i,j)~=0
            durum_no=num2str(Isq(i,j));
            olay_no=num2str(Itsigma(i,Isq(i,j)));
            if basla==0
               set_kos=strcat('q',durum_no,'.','e',olay_no);
               basla=1;
            else
               set_kos=strcat(set_kos,'+','q',durum_no,'.','e',olay_no);
            end
         end
      end
      set_kosulu(i)=cellstr(set_kos);
   end
   set_kosulu
%reset kosulu ifadesi
for i=1:Nq
   basla_rst=0;
   for j=1:length(Irsigma(i,:))
      if Irsigma(i,j)~=0
         rst_olay_no=num2str(Irsigma(i,j));
         if basla_rst==0
            reset_kos=strcat('e',rst_olay_no);
            basla_rst=1;
         else
            reset_kos=strcat(reset_kos,'+','e',rst_olay_no);
         end
      end
   end
   reset_kosulu(i)=cellstr(reset_kos);
end
reset_kosulu
```

Şekil 6.4 : Kurma ve Silme koşullarının oluşturulması

6.1.4 Durum geçiş fonksiyonuna ilişkin Kurma (Set) ve Silme (Reset) koşulları

Bölüm 5'te Tanım 5.7 ile verilen kurma koşulu ve Tanım 5.8 ile verilen silme koşulu Matlab uygulamasında string türünde değişkenler kullanılarak oluşturulabilir. Şekil 6.4'te daha önce hesaplanan Isq, Itsigma ve Irsigma değişkenlerinden faydalanarak Kurma ve Silme koşullarını karakter dizileri (character array) ve string türünde hücre dizileri (cell array) içinde oluşturmayı sağlayan program parçası görülmektedir. Bu program parçası ile *i* indisli bir duruma ilişkin Kurma (Set) koşulu "kurma_kosulu" değişkeninin *i*. bileşeni olarak oluşur, yani "kurma_kosulu(i)"dir. Benzer şekilde "reset_kosulu(i)" *i* indisli durumun Reset (Silme) koşulunu ifade eder. Kurma ve Silme koşulları (5.6) ve (5.8) ifadelerine uygun olarak elde edilmektedir.

Şekil 6.4'teki program parçasının işletilmesiyle, Şekil 4.5'teki ZS-otomat için elde edilen "kurma_kosulu" ve "silme_kosulu" değişkenlerinin içeriği Şekil 6.5'teki

Matlab komut satırı görüntüsünde verilmiştir. “q” karakteri durumlar için, “e” karakteri olaylar için kullanılmaktadır. Zamanlama ve sayma olayları da sistem olayları gibi indis numaraları ile temsil edilmektedir. Bu örnek için Şekil 6.5’te görülen e5 değişkeni zamanlama olayına, e7 değişkeni σ_2 olayına ilişkin sayma olayına karşılık gelir. “.” ve “+” karakterleri Tanım 5.7 ve Tanım 5.8’e uygun olarak sırasıyla mantıksal VE ve VEYA işlemlerine karşılık gelmektedir.

```
set_kosulu =
    'q2.e5'    'q1.e1+q5.e4'    'q2.e8'    'q2.e7'    'q2.e1+q3.e5+q4.e4'

reset_kosulu =
    'e1'    'e1+e5+e7+e8'    'e5'    'e4'    'e4'
```

Şekil 6.5 : Örnek ZS-otomat için “set_kosulu” ve “reset_kosulu” değişkenleri

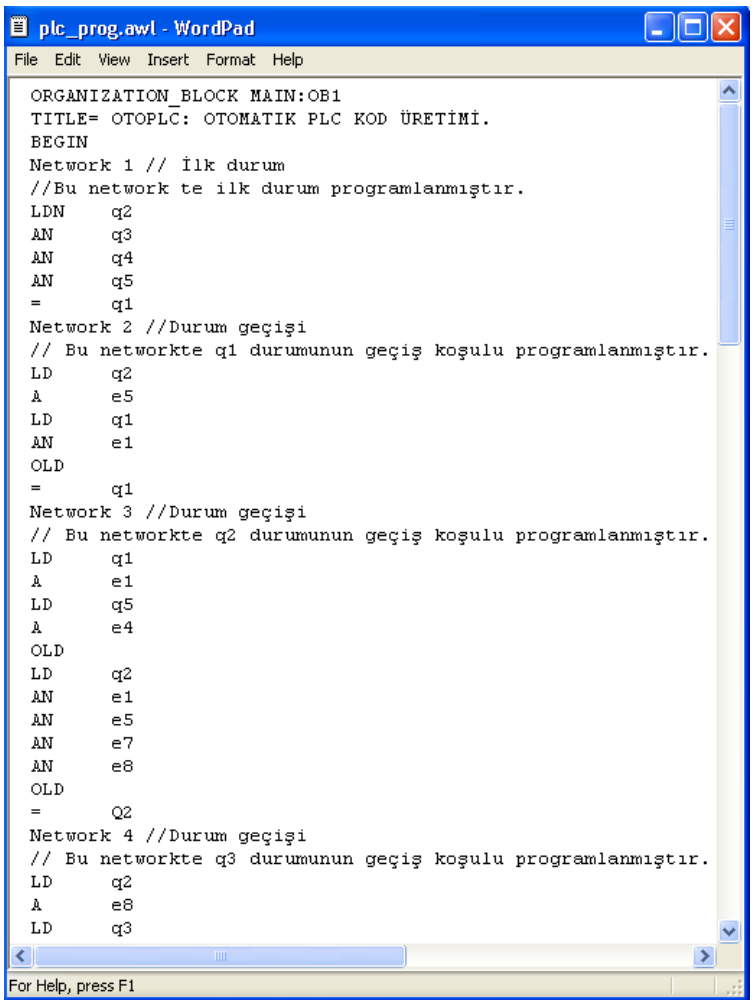
6.1.5 PLC kodunun yazdırılması

Şekil 6.5’te örneği görülen “set_kosulu” ve “reset_kosulu” değişkenleri OtoPLC uygulaması için PLC kodunun üretilmesinde kullanılmaktadır. OtoPLC uygulaması hücre dizisi şeklinde oluşturulan bu değişkenlerin her bir hücresinin içeriğindeki “.” ve “+” karakterlerine göre olay ve durum karakterlerini ayırır. PLC programı, elde edilen olay ve durumlar için Siemens S7-200 PLC’lerinin komutları kullanılarak bir dosyaya yazdırılmaktadır. Program, metin tabanlı dillerden biri olan “Yapısal Metin Dili (Structured Text Language-STL)” ile oluşturulmaktadır. Bu program, merdiven mantığındaki (Ladder Logic) karşılığına, S7-200 PLC’lerin programlama editörü Step 7-MicroWin ile kolaylıkla dönüştürülebilir.

PLC programı oluşturulurken Çığ Etkisi problemi ile ilgili olan durumlar iki ayrı değişkenle temsil edilmektedir. Gerçeklemenin tam olması için durum geçiş fonksiyonunun yanında ilk durumun, ZS-Yapısının ve çıkış fonksiyonlarının da programlanması gereklidir. ZS-Yapısına bağlı çıkış fonksiyonları daha önce belirtildiği gibi PLC’nin zamanlama ve sayma işlevlerini yerine getiren hazır Zamanlayıcı ve Sayıcı program blokları kullanılarak gerçeklenmektedir. Bunun için, (6.4) ifadesindeki örneği verilen Zamanlama ve Sayma yapısına ilişkin matris değişkenlerindeki veriler alınıp, S7-200 PLC’lerin Zamanlayıcı ve Sayıcı bloklarını kullanan PLC kodları üretilmektedir. δ duruma bağlı çıkış fonksiyonu da, (6.5) ifadesinde örneği verilen $\boldsymbol{\delta}$ vektöründeki (Şekil 6.1’de *Delta* değişkeni) veriler doğrultusunda durumlara bağlı olay çıkışlarının üretilmesiyle gerçeklenmektedir.

Yukarıda özetlenen PLC programının yazdırılmasına ilişkin Matlab program parçasına büyüklüğü nedeniyle burada yer verilememektedir. OtoPLC uygulamasının tamamı Ek B'de verilen disk içeriğinde bulunabilir.

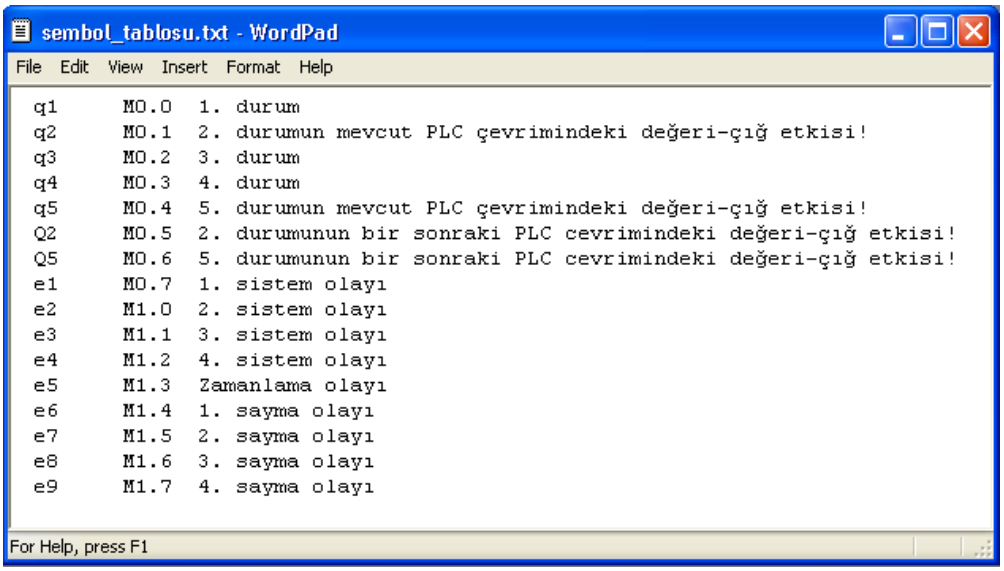
OtoPLC uygulamasının çalıştırılması ile Matlab çalışma klasörü içinde iki dosya oluşturulur. Bu dosyalardan biri PLC programına, diğeri ise sembol tablosuna karşılık gelmektedir. PLC programı, MicroWin programının içe/dışa aktar (import/export) biçimine uygun olarak "plc_prog.awl" adlı dosyanın içeriğinde oluşur. Şekil 4.5'te ele alınan örnek ZS-otomat'a ilişkin OtoPLC uygulaması tarafından üretilen PLC programının bir kısmı Şekil 6.6'da görülmektedir.



Şekil 6.6 : OtoPLC programının ürettiği PLC kodunun örnek görüntüsü

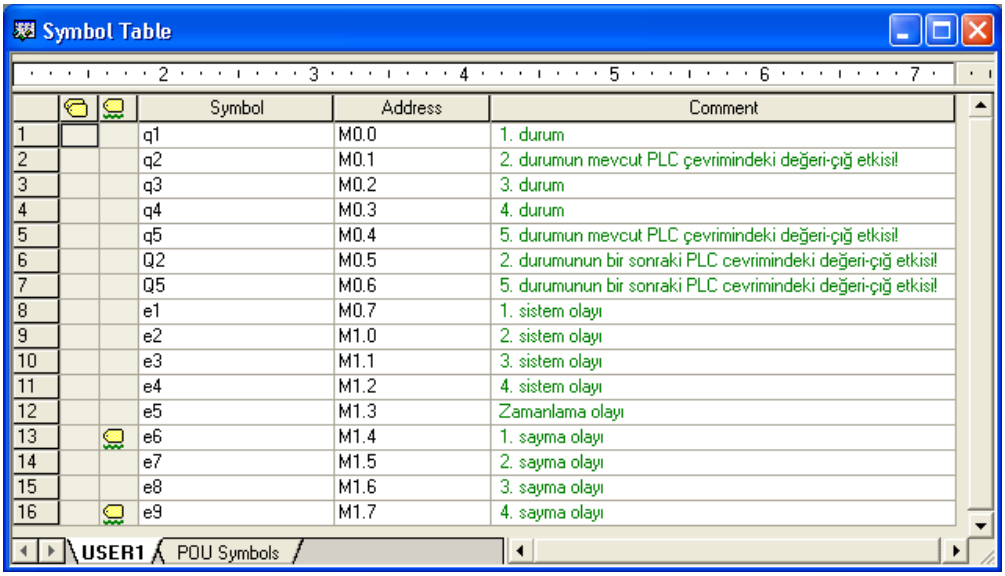
Üretilen PLC programı ana program ve bir alt programdan oluşur. Ana programda durum geçişleri, alt programda ise duruma ve ZS-Yapısına bağlı çıkış fonksiyonları programlanmıştır. Şekil 6.6'da görüldüğü gibi üretilen PLC programında PLC'ye ilişkin bellek adresleri yerine q1, e1 gibi sembolik değişkenler kullanılmıştır.

Sırasıyla durum ve olaylara karşılık gelen bu değişkenlerin PLC tarafından kullanılabilmesi için değişkenlerin PLC'nin hangi bellek alanına karşılık geldiği tanımlanmalıdır. Bu tanımlama, MicroWin'de bir sembol tablosuna değişkenlerin ve bu değişkenlere karşılık gelen adreslerin girilmesi ile yapılır. Bu işlemi kolaylaştırmak için OtoPLC uygulaması gerekli sembol tablosunu da ayrı bir metin dosyası olarak üretmektedir. Şekil 6.7'de ele alınan örnek için üretilen "sembol_tablosu.txt" adlı metin dosyasının içeriği görülmektedir.



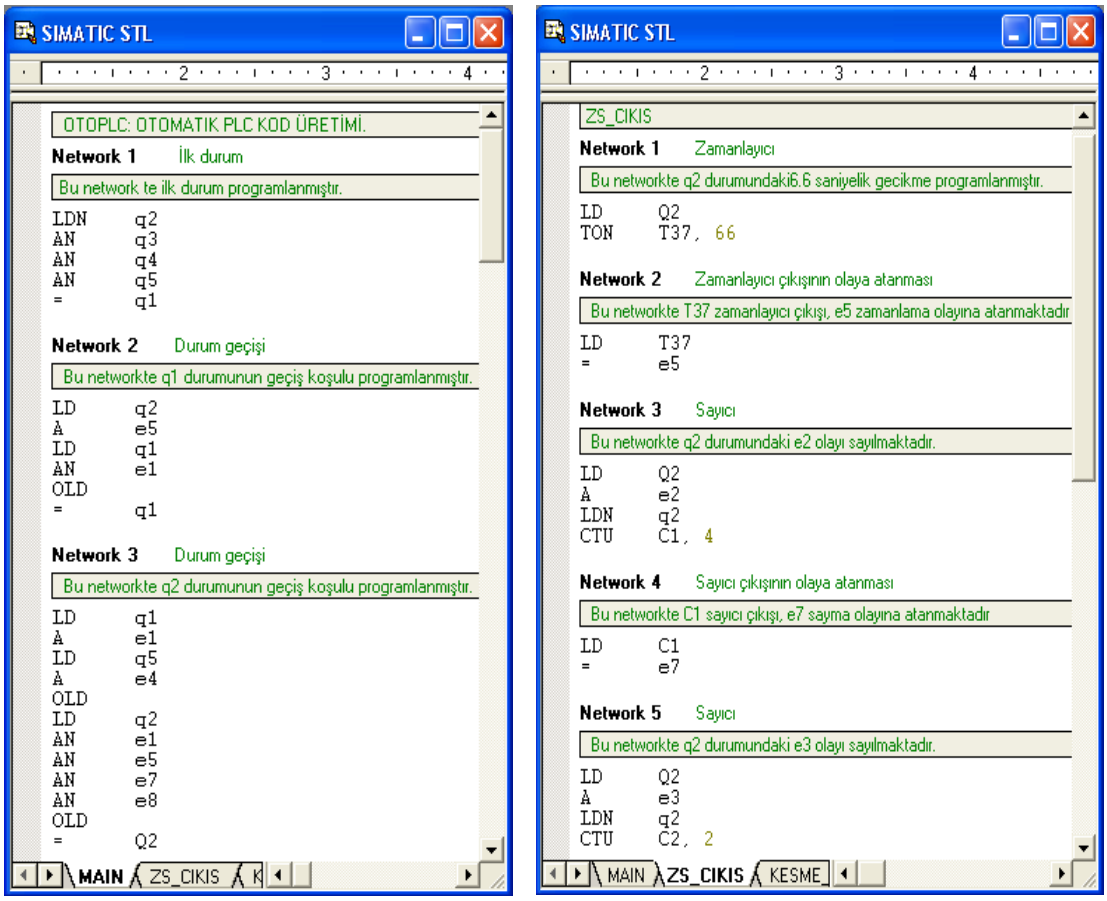
Şekil 6.7 : Üretilen sembol tablosunun metin editöründeki görüntüsü

Oluşturulan bu sembol tablosu, tamamı seçildikten sonra kopyalanarak MicroWin editörünün sembol tablosuna yapıştırılmasıyla MicroWin ortamına aktarılabilir. Şekil 6.8'de sembol tablosunun MicroWin'deki görüntüsü verilmiştir.



Şekil 6.8 : Sembol tablosunun MicroWin editöründeki görüntüsü

Sembol tablosunun bu şekilde MicroWin'de oluşturulmasından sonra, plc_prog.awl dosyasının MicroWin editörü tarafından içe aktarılması (import) ile üretilen PLC programı da MicroWin ortamına taşınmış olur. Şekil 6.9'da ele alınan örneğe ilişkin ana programın ve alt programın MicroWin editöründeki kısmi görüntüleri verilmiştir. PLC programının çalıştırılması için bu programın PLC'ye yüklenmesi gerekir.



Şekil 6.9 Ana ve alt PLC programlarının MicroWin editöründeki görüntüsü

PLC programının merdiven dilindeki görünümü MicroWin editörünün ilgili View Menüsünden "Ladder" seçilerek elde edilebilir. Elde edilen PLC programının Merdiven Dilindeki karşılığının tamamı EK A'da verilmiştir. Bu PLC programında çıkış olmayan sistem olaylarına ($\sigma_1, \sigma_2, \sigma_3$) karşılık gelen mantıksal değişkenlerin (e1, e2, e3) ilgili fiziksel işaretlerin yükselen ya da düşen kenarı ile elde edildiği varsayılmaktadır.

7. SONUÇ VE ÖNERİLER

Ayrık olay sistemlerinin karmaşıklığının artması ve yaygınlaşması analiz, tasarım ve kontrol uygulamalarında formal yöntemlerin kullanılmasını kaçınılmaz kılmıştır. Formal yöntemlerle yapılan bir tasarımın teknolojik ya da fiziksel araçlar kullanılarak hayata geçirilmesi ise gerçekleme olarak tanımlanmaktadır. Ayrık olay sistemlerinde gerçekleme konusu önemli bir çalışma alanıdır ve literatürde bu konuya yönelik çalışmaların kesin ve sistematik yöntemler tanımladığını söylemek kolay değildir.

Bu çalışmada özellikle ayrık olay sistemlerinin kontrolünde, tasarımı ifade etmek için kullanılabilecek, durum diyagramlarını temel alan bir gösterim modeli ve gerçekleme yöntemi tanımlanmış, elde edilen tanımdan hareketle otomatik PLC kod üretimi sağlayan bir Matlab uygulaması geliştirilmiştir. Tanımlanan gösterim modeli Uygulamada sıklıkla karşılaşılan davranış türlerinden zamanlama ve sayma davranışlarının ifadesini kolaylaştırmakta, aynı zamanda Programlanabilir Lojik Kontrolörlerle (PLC) gerçeklemede doğrudan kullanılabilecek bir yapıda oluşturmaktadır. Zamanlama ve sayma davranışının ifadesi standart otomat gösterimine Zamanlama ve Sayma Yapısı olarak adlandırılan bir özelliğin kazandırılması ile sağlanmaktadır. Bu özelliği ile bu otomat modeli, Zamanlama ve Sayma yapılı otomat, ya da kısaca ZS-otomat olarak adlandırılmıştır. Tanımları açısından, standart otomat gösteriminin ZS-otomatın özel bir durumu olduğu söylenebilir.

Standart otomat ya da ZS-otomat modeli ile elde edilen bir tasarımın gerçeklenmesi için iyi tanımlanmış bir gerçekleme yöntemine gereksinim duyulur. Bu gerçekleme yöntemi, tasarımın kuramsal varsayımlarının gerçekleme için kullanılan fiziksel ortamın doğasıyla örtüşmemesi nedeniyle istenenden farklı davranışların elde edilmesine neden olmamalıdır. Uygulamada bu anlamda sıklıkla karşılaşılan bir gerçekleme sorunu *çığ etkisi* olarak adlandırılır ve otomat gösterimindeki belirli yapısal bir özelliğin PLC'ler ile gerçeklemede hatalı sonuçlar vermesi ile ilgilidir.

Literatürde bu sorun için sistematik olmaktan uzak, bazı uygulamalarında ise doğru bir gerçeklemenin elde edilmesinin imkânsız olduğu yöntemler önerilmiştir.

Bu çalışmada önerilen ZS-otomat için, çığ etkisi problemi olmayan, sistematik bir gerçekleme yöntemi tanımlanmıştır. Yöntem uygulamacı tarafından doğrudan kullanılarak tasarıma karşılık gelen PLC programının oluşturulması için kullanılabilmektedir. Tamamen matematiksel ve sistematik bir şekilde ifade edilmesine rağmen, bu yöntem sezgisel olarak kolay uygulanabilir, elde edilen PLC kodu ise kolay anlaşılabilir olma özelliğindedir. Tanımlanan yöntem uygulandığında, durum geçişlerine ilişkin koşullar elde edilen PLC programında açıkça görülmektedir ve her bir duruma ilişkin geçiş koşulları için merdiven mantığı programında tek bir basamak yeterli olmaktadır.

Tanımlanan gerçekleme yöntemi, programlanabilir olma özelliğine de sahiptir. Bu özellik tasarıma karşılık gelen PLC programının otomatik olarak üretilmesi için bir Matlab uygulamasının geliştirilmesinde kullanılmıştır. Matlab uygulaması, tasarım sonucu elde edilen ZS-otomata ilişkin tanımları (durum geçiş fonksiyonu, ZS-Yapısı, çıkış fonksiyonu ve ilk durum gibi) matris, vektör ve skaler değişkenler şeklinde giriş olarak alıp, gerçeklemeye karşılık gelen PLC programını bir metin dosyasına yazdırmaktadır. Matlab uygulaması bu haliyle Siemens Simatic S7-200 PLC'ler için kod üretmektedir. Farklı PLC modellerinin, ya da IEC 61131-3 standardının tanımladığı kod kümelerinin kullanılması ile kolaylıkla diğer PLC modelleri için de uyarlanabilir. Matlab uygulaması ile ilgili gelecekte yapılabilecek bir geliştirme, kullanıcının tasarladığı ZS-otomatı ve buna ilişkin parametreleri grafik ara yüzler kullanarak tanıtmasıyla sağlanabilir. Bu şekilde, tasarlanan ZS-otomata ilişkin durum geçiş fonksiyonu, zamanlama ve sayma yapısı gibi verilerin matris değişkenleri ile ifade edilmesine gerek kalmadan doğrudan PLC ile gerçeklenmesi mümkün olabilir.

KAYNAKLAR

[1] **Ramadge, P.J. and Wonham, W. M**., 1987. Supervisory control of a class of discrete event processes, *SIAM Journal of Control and Optimization*, **25 (1)**, 206-230.

[2] **Ramadge, P.J. and Wonham, W. M**., 1987. On the supremal controllable sublanguage of a given language, *SIAM Journal of Control and Optimization*, **25 (3)**, 637-659.

[3] **Chen, E. and Lafortune, S**. 1991. Dealing with blocking in supervisory control of discrete-event systems, *IEEE Transactions on Automatic Control*, **36,** 724-735.

[4] **Ramadge, P.J. and Wonham, W. M**., 1988. Modular supervisory control of discrete event systems, *Mathematics of control signals and systems*, **1 (1)**, 13-30.

[5] **Queiroz, M.H. de and Cury, J.E.R**., 2000. Modular control of composed systems, *American Control Conference*, Chicago, USA, June 28-30

[6] **Queiroz, M.H. de and Cury, J.E.R**., 2000. Modular supervisory control of large scale discrete-event systems. *5th Int. Workshop on Discrete Event Systems (WODES)*, Ghent, Belgium.

[7] **Rohloff, K. and Lafortune, S**., 2003. Supervisor existence for modular discrete-event systems, *2nd IFAC Conference on Control Systems Design*, Bratislava, Slovakia.

[8] **Lin, F. and Wonham, W. M**., 1988. Decentralized supervisory control of discrete-event systems. *Information Sciences*, **44**,199–224.

[9] **Jiang, S., Chandra, V. and Kumar, R**., 2001. Decentralized control of discrete event systems with multiple local specifications. *American Control Conference*, Arlington, Virginia, USA, June 25-27, 959–964.

[10] **Lin, F. and Wonham, W. M**., 1991. Verification of nonblocking in decentralized supervision. *Control-Theory and Advanced Technology*, **7(1),** 223–232.

[11] **Yoo, T.-S. and Lafortune, S**., 2004. Decentralized supervisory control with conditional decisions: Supervisor existence. *IEEE Transactions on Automatic Control*, **49,** 1886-1904.

[12] **Zhong, H. and Wonham, W. M**., 1990. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, **35,** 1125-1134.

[13] **Leduc, R.J., Brandin, B.A., Wonham, W. M. and Lawford, M**., 2001. Hierarchical interface-based supervisory control: Serial case. *40th IEEE Conference on Decision and Control,* Oriando, Florida USA, 4116-4121.

[14] **Leduc, R.J**., 2002. Hierarchical interface-based supervisory control. *PhD thesis*, University of Toronto, Toronto, Canada.

[15] **Rohloff, K. and Lafortune, S**., 2006. The Verification and Control of Interacting Similar Discrete-Event Systems. *SIAM Journal on Control and Optimization,* **45(2)**, 634-667.

[16] **Uzam M. and Wonham W.M**., 2006. A Hybrid Approach to Supervisory Control of Discrete Event Systems Coupling RW Supervisors to Petri Nets, *International Journal of Advanced Manufacturing Technology*, **28**, 747-760.

[17] **Balemi S., Hoffmann G.J., Gyugyi P., Wong-Toi H., and Franklin G.F**., 1993. Supervisory control of a rapid thermal multiprocessor, *IEEE Transactions on Automatic Control*, **38 (7)**, 1040-1059.

[18] **Brandin, B. A**., 1996. The real-time supervisory control of an experimental manufacturing cell, *IEEE Transactions on Robotics and Automation*, **12 (1)**, 1-14.

[19] **Leduc R. J**., 1996. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation Perspective. *Master's thesis*, University of Toronto, Toronto, Canada.

[20] **Music, G. and Matko, D**., 2001, Discrete Event Control Theory Applied to PLC Programming. *9th Mediterranean Conference on Control and Automation*, Dubrovnik, Croatia.

[21] **Chandra, V., Huang, Z., Kumar, R**., 2003. Automated control synthesis for an assembly line using discrete event system control theory, *IEEE*

Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, (**33)**, 284-289.

[22] **Fabian M. and Hellgren A**., 1998. PLC-based implementation of supervisory control for discrete event systems, *37th IEEE conference on Decision & Control*, Tampa, Florida, USA

[23] **Queiroz, M.H. de and Cury, J.E.R**., 2002. Synthesis and implementation of local modular supervisory control for a manufacturing cell. *6th Int. Workshop on Discrete Event Systems (WODES)*, Zaragoza, Spain.

[24] **Vieira, A.D., Cury, J.E.R., de Queiroz, M.H**., 2006, A Model for PLC Implementation of Supervisory Control of Discrete Event Systems, *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Prague, Czech Republic.

[25] **Hellgren, A., Lennartson, B. and Fabian, M**., 2002. Modelling and PLC-based Implementation of Modular Supervisory Control. *6th International Workshop on Discrete Event Systems (WODES),* Zaragoza, Spain.

[26] **Hellgren, A., Fabian, M., Lennartson, B**., 2001. On the Execution of Discrete Event Systems as Sequential Function Charts *IEEE Conference on Control Applications, Mexico City, Mexico*.

[27] **Zaytoon, J. and Carré-Ménétrier, V**., 2001. Synthesis of control implementation for discrete manufacturing systems. *International Journal of Production Research*, **39**, 329-345.

[28] **M. Uzam, M. and Jones, A.H**., 1998. Discrete Event Control System Design Using Automation Petri Nets and Their Ladder Diagram Implementation, *International Journal of Advanced Manufacturing Technology*, **14 (10),** 716-728.

[29] **Uzam, M., Jones, A.H. and Yücel, İ**., 2000. Using a Petri-Net-Based Approach for the Real-Time Control of an Experimental Manufacturing System, *International Journal of Advanced Manufacturing Technology*, **16**, 498-515.

[30] **Cansever, G. and Kucukdemiral, I. B**., 2006. "A new approach to supervisor design with sequential control petri-net using minimization technique for discrete event system," *International Journal of Advanced Manufacturing Technology*, **29**, 1267–1277.

[31] **Hasdemir İ. T., Kurtulan S. and Gören L**., 2003. Supervisory control of a pneumatic system using PLC. *Third International Conference on Electrical and Electronics Engineering (ELECO)*, Bursa, Turkey.

[32] **Hasdemir İ.T., Kurtulan S. and Gören L**., 2004. Implementation of local modular supervisory control for a pneumatic system using PLC. *7th International Workshop on Discrete Event Systems (WODES)*, Reims, France.

[33] **Hasdemir İ.T., Kurtulan S. and Gören L**., 2008. An implementation methodology for supervisory control theory. *International Journal of Advanced Manufacturing Technology*. **36,** 373-385.

[34] **Hasdemir İ.T. and Kurtulan S**., 2006. Automatic PLC code generation using Matlab. *3rd IFAC Workshop on Discrete Event System Design,* Rydzyna, Poland, 131-136.

[35] **Hasdemir İ.T., Kurtulan S**., **ve Gören, L**., 2006. Ayrık olay sistem kontrolünde üstdenetleyicilerin PLC ile gerçeklenmesi. *Otomatik Kontrol Ulusal Toplantısı (TOK)*. Ankara

[36] **Cassandras, G.C. and Lafortune, S**., 1999. Introduction to discrete event systems. Kluwer Academic Publishers, Massachusetts, USA.

[37] **Hopcroft, J.E. and Ullman, J.D**., 1979. Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Massachusetts.

[38] **Murata, T**., 1989. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, **77 (4)**, 541-580.

[39] **Peterson, J. L**., 1981. Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs, N.J., USA.

[40] **Alur, R. and Dill, D. L**., 1994. A theory of timed automata, *Theoretical Computer Science*, **126**, 183-235.

[41] **Dierks, Henning**., 2001. PLC-automata: a new class of implementable real-time automata, *Theoretical Computer Science*, **253**, 61-93.

[42] **Frey, G. and Litz, L**., 2000. Formal methods in PLC programming, *IEEE Conference on Systems Man and Cybernetics (SMC)*, Nashville, USA

[43] **Kurtulan, S**., 2007. Endüstriyel kumanda sistemleri, Nobel Yayın Dağıtım, İstanbul.

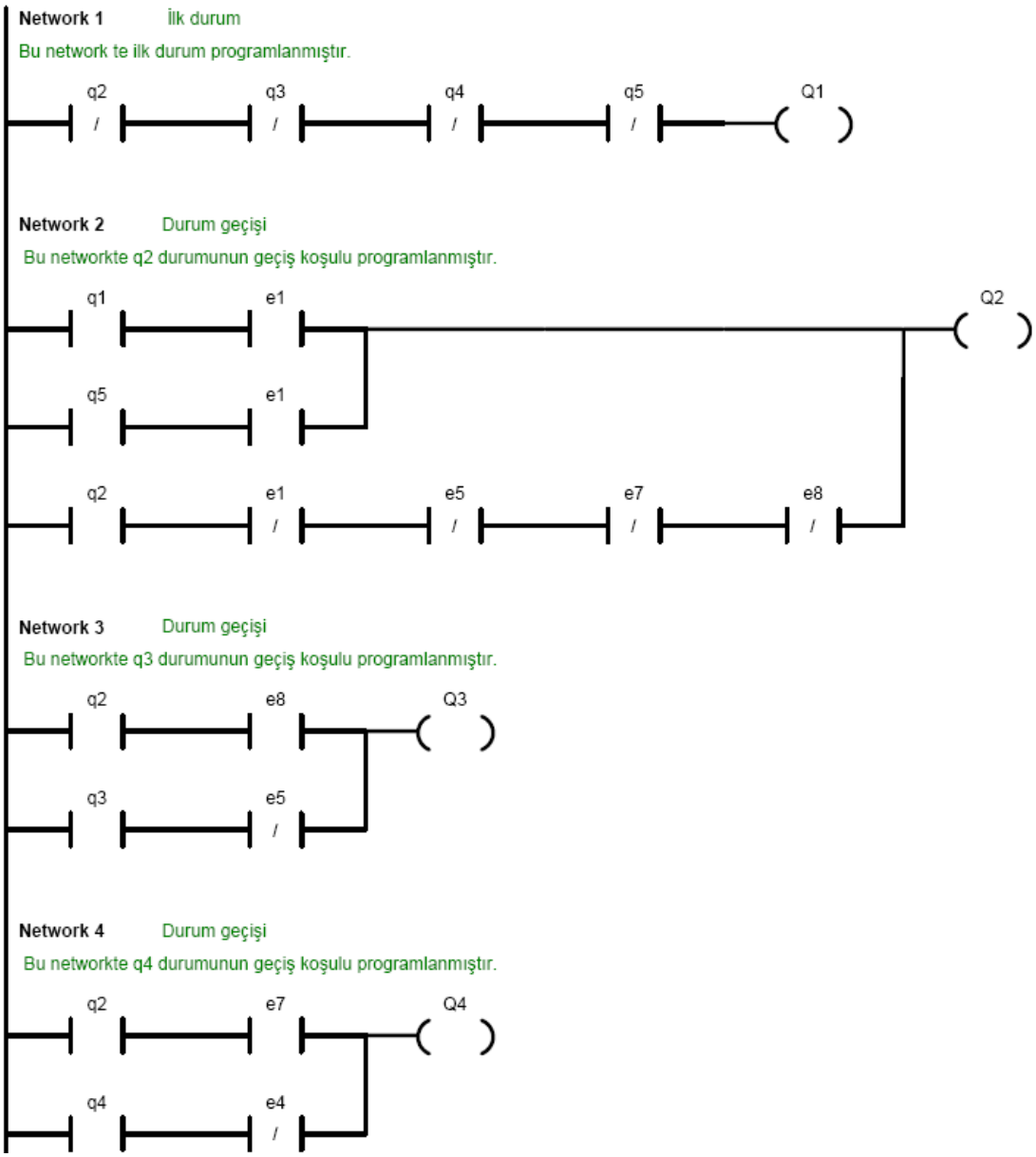
[44] **Bolton, W**., 1997. Programmable Logic Controllers: An Introduction, Butterworth-Heinemann.

[45] **Siemens** 2004. Simatic S7-200 Programmable Controller System Manual.

[46] **IEC 61131-3**, 2003. Programmable controllers-Part 3:Programming languages. International Electrotechnical Commission.

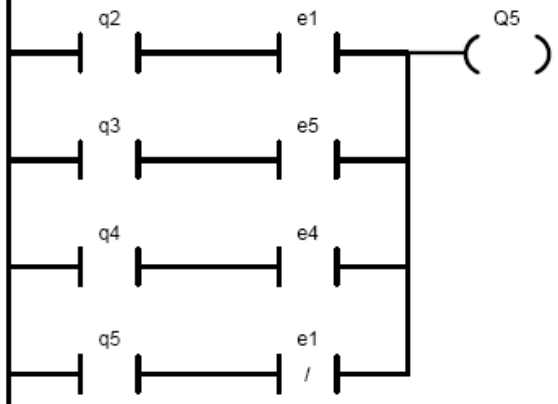
EK A. OTOPLC UYGULAMASI İLE ELDE EDİLEN PLC PROGRAMI

A.1. Ana Program

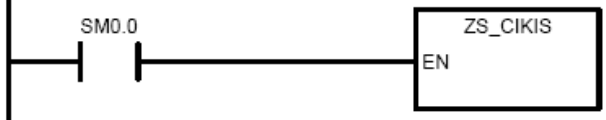


Network 5 Durum geçişi

Bu networkte q5 durumunun geçiş koşulu programlanmıştır.

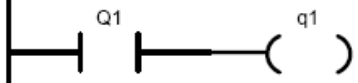


Network 6 ZS_CIKIS Alt Programinin Cagirilmasi



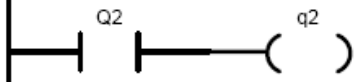
Network 7 Durum senkronizasyonu

Bu networkte Q1 değişkeni, q1 değişkenine atanmaktadır.



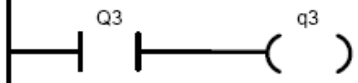
Network 8 Durum senkronizasyonu

Bu networkte Q2 değişkeni, q2 değişkenine atanmaktadır.



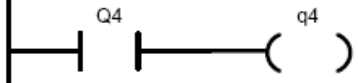
Network 9 Durum senkronizasyonu

Bu networkte Q3 değişkeni, q3 değişkenine atanmaktadır.



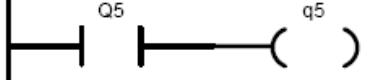
Network 10 Durum senkronizasyonu

Bu networkte Q4 değişkeni, q4 değişkenine atanmaktadır.

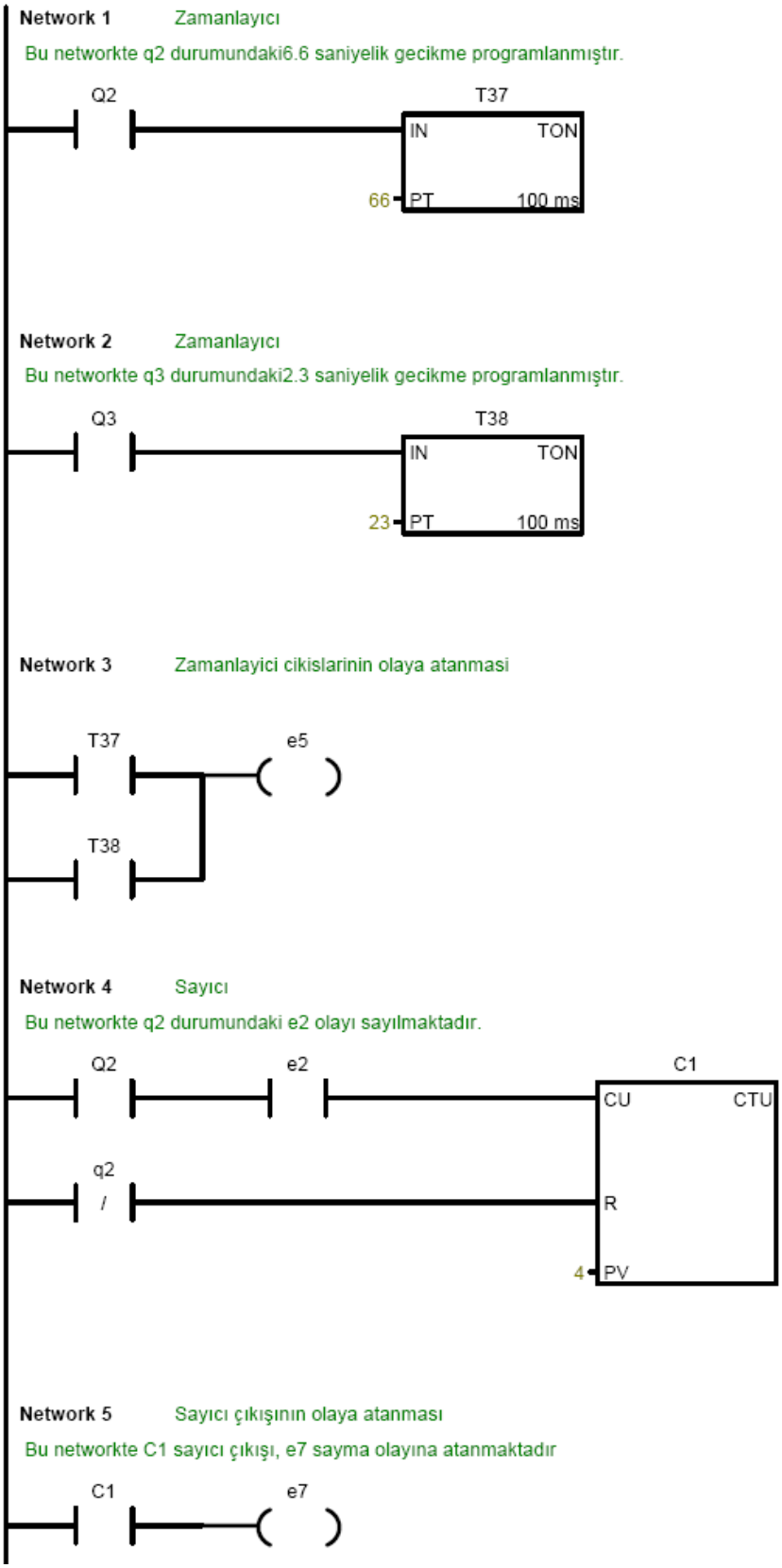


Network 11 Durum senkronizasyonu

Bu networkte Q5 değişkeni, q5 değişkenine atanmaktadır.

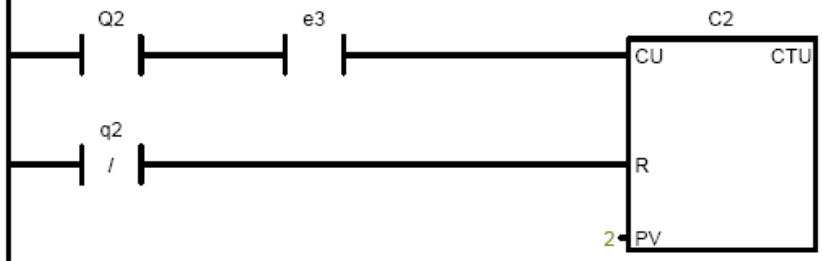


A.2. Alt Program



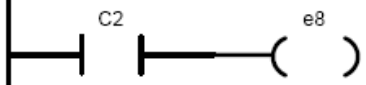
Network 6 Sayıcı

Bu networkte q2 durumundaki e3 olayı sayılmaktadır.



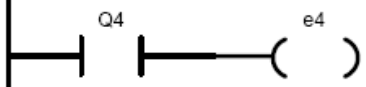
Network 7 Sayıcı çıkışının olaya atanması

Bu networkte C2 sayıcı çıkışı, e8 sayma olayına atanmaktadır



Network 8 Duruma bağlı çıkış fonksiyonunun gerçekleştirilmesi

Bu networkte q4 durumuna ilişkin, e4 çıkış olayı üretilmektedir.



ÖZGEÇMİŞ

Ad Soyad:	İbrahim Tolga HASDEMİR
Doğum Yeri ve Tarihi:	Malatya, 06.09.1976
Lisans Üniversite:	İstanbul Teknik Üniversitesi, Elektrik Mühendisliği
Y. Lisans Üniversite:	İstanbul Teknik Üniversitesi, Kontrol ve Bilgisayar Mühendisliği

Yayın Listesi:

Hasdemir İ. T., Kurtulan S. and Gören L., 2003. Supervisory control of a pneumatic system using PLC. Third International Conference on Electrical and Electronics Engineering (ELECO), Bursa, Turkey.

Hasdemir İ.T., Kurtulan S. and Gören L., 2004. Implementation of local modular supervisory control for a pneumatic system using PLC. 7th International Workshop on Discrete Event Systems (WODES), Reims, France.

Hasdemir İ.T., Kurtulan S. and Gören L., 2008. An implementation methodology for supervisory control theory. *International Journal of Advanced Manufacturing Technology*. **36,** 373-385.

Hasdemir İ.T. and Kurtulan S., 2006. Automatic PLC code generation using Matlab. *3rd IFAC Workshop on* Discrete *Event System Design,* Rydzyna, Poland, 131-136.

Hasdemir İ.T., Kurtulan S., ve Gören, L., 2006. Ayrık olay sistem kontrolünde üstdenetleyicilerin PLC ile gerçeklenmesi. *Otomatik Kontrol Ulusal Toplantısı (TOK)*. Ankara