İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ANKLAŞMAN SİSTEMİ YAZILIM MODELİNİN MODEL SINAMASININ YAPILMASI

YÜKSEK LİSANS TEZİ

Davut POLAT

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

MAYIS 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ANKLAŞMAN SİSTEMİ YAZILIM MODELİNİN MODEL SINAMASININ YAPILMASI

YÜKSEK LİSANS TEZİ

Davut POLAT
(504121553)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Tolga OVATMAN

MAYIS 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504121553 numaralı Yüksek Lisans Öğrencisi **Davut POLAT**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ANKLAŞMAN SİSTEMİ YAZILIM MODELİNİN MODEL SINAMASININ YAPILMASI"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Yrd. Doç. Dr. Tolga OVATMAN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Yrd. Doç. Dr. Ahmet Cüneyt TANTUĞ** İstanbul Teknik Üniversitesi	.....
	Yrd. Doç. Dr. Muhammed Ali AYDIN İstanbul Üniversitesi	.....

Teslim Tarihi : **4 Mayıs 2015**
Savunma Tarihi : **28 Mayıs 2015**

Eşim Ayşe Canan ve oğlum Süleyman'a

ÖNSÖZ

Bu tez çalışmasının gerçekleşmesini sağlayan ve yardımlarını hiç esirgemeyen değerli hocam Yrd. Doç. Dr. Tolga OVATMAN'a teşekkürlerimi sunarım. Bu tez çalışması 113E272 nolu ve PLC Tabanlı Demiryolu Anklaşman Yazılımlarının Model Sınama Kullanılarak Doğrulanması isimli Tübitak 1001 projesi ile ortak çalışma sonucu yapılmıştır. Tez çalışmamdaki katkılarından dolayı 2210- Yurt İçi Yüksek Lisans Burs Programı ile yüksek lisans çalışmamı destekleyen TÜBİTAK Bilim İnsanı Destekleme Daire Başkanlık'ına ve İTÜ Endüstriyel Otomasyon Laboratuvarı çalışanlarına ayrıca teşekkür ederim. Tez çalışmam boyunca sabır ve desteklerinden dolayı eşim Ayşe Canan, oğlum Süleyman'a teşekkür ederim.

Mayıs 2015 Davut POLAT

İÇİNDEKİLER

Sayfa

KISALTMALAR

ANTLR : Another Tool For Language Recognition
CTL : Computational Temporal Logic
FBD : Function Block Diagram
IEC : International Electrotechnical Commission
IL : Instruction List
LD : Ladder Diagram
LTL : Linear Temporal Logic
PLC : Programmable Logic Control
PNLF : Petri Net Linear Form
SAT : Satisfiability
ST : Structured Text
STL : Security Integration Level
TAPN : Timed-Arc Petri Net
TCDD : Türkiye Cumhuriyeti Devlet Demiryolları
TON : Timer On Delay
TPN : Timed Petri Nets
UDSP : Ulusal Demiryolu Sinyalizasyon Projesi
XML : Extensible Markup Language

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

ANKLAŞMAN SİSTEMİ YAZILIM MODELİNİN MODEL SINAMASININ YAPILMASI

ÖZET

Tarihsel gelişim süreci içerisinde demiryolu trafiğinin düzenlenmesi ilk zamanlarda tamamen insan eliyle yürütülürken günümüzde elektronik sistemler tarafından otomatik olarak gerçeklenmektedir. Gelişen teknoloji neticesinde günümüzde özel olarak üretilen adanmış donanımlar üzerinde koşan yazılımlar yerini daha modüler ve esnek biçimde kullanılabilen ve daha düşük maliyetli PLC(Programmable Logic Control) tabanlı sistemlere bırakmaktadır. Anklaşman sistemlerinin SIL(Security Integrity Level) gibi ciddi güvenlik standartlarını sağlaması beklenmekte ve bu amaçla ciddi sınama eforu harcanmaktadır.

Otomatik anklaşman sistemleri, günümüz demiryolu sistemlerinin işleyişinin düzenlenmesinin güvenli bir biçimde yapılmasını sağlayan en önemli bileşenlerdir. Bu sebeple, karmaşık hatlara ve yoğun tren trafiğine sahip istasyonlarda kullanılan anklaşman sistemlerinin doğru çalışacak biçimde üretilmesi büyük önem taşımaktadır. Bu tez çalışmasında, ülkemizdeki bir tren istasyonunda kullanılmakta olan PLC tabanlı bir anklaşman sistemi yazılım prototipinin biçimsel bir yöntem olan model sınama teknikleri kullanılarak doğrulanması amaçlanmaktadır.

Anklaşman sistemlerinin sınanması farklı düzeylerde ve çeşitli yaklaşımlar kullanılarak gerçekleştirilmektedir. Bu düzeyler anklaşman yazılımı bileşenlerinin tabi tutulduğu birim testlerinden, gerçek sistemin bilfiil denendiği saha testlerine kadar farklılık gösterebilmektedir. Sınamaların gerçekleştirilmesindeki temel kaygı sistem çalışması sırasında oluşabilecek tüm olası durumların sınanmasının zaman ve bellek gibi mali kısıtlar nedeniyle her zaman mümkün olmamasının yanında sözü edilen olasılıkların oluşturulma işlemi de bileşen-durum sayısının büyüklüğü nedeniyle kendi içinde zorluklara sahiptir. Bu zorlukların aşılmasında çeşitli otomatik sınama yöntemleri ve durum üretim yöntemleri kullanılmakla birlikte yapılan sınamaların kapsadığı durum sayısı çoğu zaman yetersiz kalmaktadır.

Anklaşman sistemleri gibi gerçek zamanlı sistemlerin sınanmasında kullanılan biçimsel yaklaşımlardan bir tanesi de model sınama yöntemleridir. Model sınamada sınanacak sistem bileşenlerinin soyut durum tabanlı modelleri oluşturularak otomatik bir biçimde sistem bileşenlerinin olası bütün durum geçiş senaryoları oluşturularak biçimsel bir şekilde ifade edilmiş sistem isterlerinin oluşturulan sistemde geçerli olup olmadığı kontrol edilmektedir. Model sınamanın avantajı üretilen soyut modellerin çeşitli detaylarda oluşturularak sistemin odaklanan belirli özellikleri üzerinde sınamalar yapılabilmesidir. Daha da önemlisi model sınama sistem çalışması esnasında oluşabilecek olası tüm durumları gözden geçirdiği için elde edilen sonuçlar modellenen sistem için "garanti" sonuçlardır ve bütün durum uzayını kapsarlar.

Yapılan tez çalışmasında, PLC tabanlı oluşturulmuş anklaşman yazılımlarının durum tabanlı modellerinin oluşturulup, model sınama teknikleri ile doğrulanması için

yöntemler geliştirilmesi amaçlanmaktadır. Geliştirilecek yöntemler 2009-2012 yılları arasında İTÜ ile TÜBİTAK BİLGEM ortaklığıyla gerçekleştirilmiş UDSP(Ulusal Demiryolu Sinyalizasyon Projesi) kapsamında İTÜ tarafından üretilen PLC yazılım prototipleri üzerinde denenerek geliştirilmiştir. Tez amacı olarak, otomatik anklaşman yazılımları için güvenlik garantileri verebilen bir sınama yöntemi geliştirmektir. Bu tür sınama yöntemi, sistem geliştirilmeden model üzerinde ortaya çıkaracağı kusurlar ile mali ve sistem geliştirildikten sonra ortaya çıkaracağı kusurlar ile sistem güvenliği açısından önemli faydalar sağlayacaktır.

MODEL VERIFICATION OF INTERLOCKING SOFTWARE SYSTEM

SUMMARY

While, in the early days of railway interlocking, the process was achieved by total human control, today's systems are almost completely controlled by automatic electronic system and software. Moreover the dedicated hardware and specialized software being used in developing such systems is being replaced by PLC(Programmable Logic Control) based systems because of the modularity and cost effectiveness of those systems. Like every interlocking system PLC based interlocking systems also need to be heavily tested and verified to satisfy safety properties in order to meet certain safety standards like SIL(Security Integration Level).

Automatic interlocking systems are the crucial components in the safe orchestration process of today's railway systems. Especially when it comes to regulating complicated train traffic in a station, the process becomes more important to ensure safety in the station. In the proposed project, verification of an interlocking system software being used in a local train station is going to be performed using model checking techniques.

Verification of interlocking systems can be achieved at various levels and various verification techniques. These levels may change from unit tests applied on basic system components to the field tests, which are applied on real hardware and on actual train stations. Biggest concern on such tests are not only being unable to test every produced scenario because of time and memory constraints but also being unable to produce every meaningful testing scenario because of the unmanageable number and state space of system components. To overcome such difficulties automatic test generation tools are often used however even those tools cannot produce satisfactory results sometimes.

Model checking is one of the formal methods that can be used in verification of such real-time systems. In model checking, abstract state-based models of system components are built by system designers and feed into the model checker. Model checker then produces every possible state change interleavings of those models and verify certain properties explicitly stated by means of formal specification and mostly temporal logic. One important advantage of model checking is being able to produce abstract models in various levels of detail and focus on different properties of the system in distinct verification phases. A more important advantage is providing guarantees based on the model under verification since the model checker checks every possible execution of the system under consideration.

In this thesis, the aim is to develop techniques to produce abstract models from PLC software of railway interlocking systems and verify produced models using model checking. Techniques that was developed will be tested on the PLC software prototypes of the interlocking system produced by National Railway Signalization

Project performed by İTÜ and TÜBİTAK BİLGEM between 2009-2012. The project aims to produce a model checking based verification system that can provide safety guarantees on the domain of PLC based interlocking which is expected to be used wider in the national railway systems. Such a system is going to provide significant benefits both in terms of cost by identifying possible defects before the interlocking software is produced and also in terms of safety by detecting defects after the interlocking system is developed.

PLC software is generally used in critical systems that work as parallel and/or real-time fashion. PLC programming languages, having a simpler syntax than modern programming languages, are characteristically closer to machine instruction level. These properties frequently make them subject to the verification and model checking studies. Model checking studies can be applied over the elements (i.e. signal lights, track circuits, interlocks) of an interlocking system is presented where PLC software under verification controls and orchestrates the behavior of interlocking elements.

Presented study is applied on text based PNLF(Petri Net Linear Form) representations of Petri Net models of PLC programs since Petri Nets can be used to model PLC software independent of the programming language used. By representing the PLC programs as Petri Nets and parsing PNLF documents that represent those Petri Nets, it became possible to transform the PLC programs to model checker language by the aid of computer programs.

PLC programs develop via FBD(Function Block Diagram) programming language is transformed into Petri Nets. Since PNLF is a text based representation, it is used in order to process a graphical representation of Petri Net by computer. For this manner a grammer was designed for defining rules of PNLF. Those rules are processed with Java Programming language by using ANTLR(Another Tool For Language Recognition) open source library. In the rules, both petri net elements(place,transitions, tokens, etc) and some interlocking elements(signals) are represented.

ANTLR is used for parsing a PNLF statement and extract all elements by the rules defined in grammer. Extracted elements are used to generate UPPAAL models for each PLC software elements(function blocks in FBD). Model checking queries also generated for each UPPAAL models. Since some PLC software elements(such as Timer On Delay) can not be transformed into petri nets, models which includes such elements are modified manually by adding some places, transitions and clock variables into UPPAAL model, but their queries are generated automatically. this modification affects state space negatively. After all UPPAAL models are automatically generated for each PLC software elements, high-level system definition is also automatically generated.

After the transformation process, model checking is applied on each different interlocking system element by regarding real-time and relative-time properties. Moreover, model checking is also applied on a basic interlocking system where many interlocking system elements work in coordination. While model checking process, state explosion occured for many models. Also some limitation in UPPAAL deadlock occured for all models. Because of the state space explosion and deadlock in whole interlocking system verification a number of abstraction techniques are applied to overcome problem. Those abstraction include a nondeterministic automata and a dual automata for models generated for each PLC software elements. After applying these

abstraction, model checking become possible for some models, but for other state space explosion continues. Model checking process is done in limited assumptions.

1. GİRİŞ

Elektronik demiryolu anklaşman sistemleri günümüzde demiryolu sistemlerinin bileşenlerinin yönetilerek demiryollarında trafiğin düzenlenmesinde insan gücünün kullanılmasını azaltmayı amaçlayan sistemlerdir. Demiryolu sistemlerinde pek çok bileşen (sinyaller, makaslar ve ray devreleri) bir arada çalışarak demiryolu üzerinde hareket eden trenlerin trafiğini aksatılmadan ve güvenli bir şekilde düzenlenmesini amaçlar. Geçmişte demiryolu trafiği kontrolu, istasyonlarda veya kritik geçişlerin olduğu noktalarda bir kişi tarafından yerinde müdahele edilerek, makas değiştirilip ya da sinyal yakıp söndürülerek yapılmaktaydı. Günümüzde insan müdahalesi ortadan kaldırılıp bu kontroller bilgisayar sistemleri yardımıyla otomatize edilmiştir.

Ülkemizde halen demiryolu trafiği kontrolü %80 oranında elle müdahelerle yapılmaktadır. Günümüzde tren hız limitlerinin git gide artması ve tren sayısının çokluğu sebebiyle bu yöntemin hataya açık olmasına sebep olmaktadır. Bu durumda çok büyük güvenlik sorunları ortaya çıkmaktadır. TCDD(Türkiye Cumhuriyeti Devlet Demiryolları) bu güvenlik açıklarını ortadan kaldırmak için çalışmalar yürütmektedir.

Demiryollarındaki kontrolleri bilgisayar sistemi ile otomatize etmek için PLC sistemleri kullanılmaktadır. PLC'ler kolay programlanabilir ve belirli bir iş için çok yüksek bir verimle adanmış şekilde çalışabildikleri için bir çok endüstri tarafından tercih edilmektedir.

Elektronik Anklaşman sistemlerinin sahada kullanılmadan önce yoğun bir test sürecinden geçmesi gerekmektedir. Bu test süresi kapsamında, kullanılan elektronik elemanların ve yazılımların birim testleri ve bu elemanların birbirleri ile uyumlu bir şekilde çalıştığını gösteren entegrasyon testleri olmalıdır. Test senaryoları hazırlanırken, sistemin çalıştığında olabilecek tüm durumların uzayını kapsaması gerekmektedir. Fakat bu durum uzayı çok büyük olduğundan, yapılan testler bu durum uzayının bir alt kümesi üzerinde gerçeklenmektedir. Test senaryolarında kapsanan durumlarda, güvenlik ve çalışabilirlik gibi öncelikli durumlar olması için çalışmalar yapılmaktadır.

Son yıllarda yapılan çalışmalarda yazılım sistemlerinde, sistem tasarım aşamasında yapılan hataların büyük maliyetlere sebep olduğu ortaya çıkmıştır. Tasarım aşamasındaki hatalar insan gücüyle gözden geçirilerek yapılabileceği gibi biçimsel yöntemler kullanılarak otomatik olarak da yapılabilir. Model sınama yöntemi, tüm olasılık uzayını ele alarak tasarımları otomatik olarak gözden geçirebildiği için çok uygun bir yöntemdir.

1.1 Tezin Amacı

Yapılan bu tez çalışmasının konusu, yukarıda açıklanan anklaşman sistemi yazılımlarının tasarımlarının model sınama yöntemleri ile doğrulanmasına olanak verebilecek tekniklerin üretilmesidir. Model sınama teknikleri, günümüz sistemlerinin artan işlem gücü yetenekleri ile gün geçtikçe uygulanabilirliklerini arttırmaktadır. Ortaya çıktığı günden bu yana gerçek zamanlı yazılımların doğrulanmasında kullanılan bu yöntemin en etkin olarak kullanıldığı alanlardan biri de, bu tür anklaşman sistemi yazılımlarıdır. Fakat, henüz yeni bir uygulama alanı olması nedeniyle PLC tabanlı anklaşman sistemleri ile gerçekleştirilen uygulamaların sayısı çok azdır. Bu durum hem araştırma olanakları açısından hem de yöntemin uygulanması sonucu ortaya çıkaracağı faydalar açısından büyük bir potansiyele sahiptir. Konuyla ilgili literatürde bulunan çalışmalar, çalışmada kullanılan yöntemler ve alınan sonuçlar ilgili bölümlerde açıklanacaktır.

1.2 Literatür Araştırması

Tez çalışması sırasında yapılan literatür araştırmasında, geçmişte anklaşman sistemlerin sınanması ve doğrulanması ile ilgili birçok çalışma incelendi. Fakat PLC tabanlı sistemlerin sınanması ile ilgili literatürde oldukça az çalışma bulunmaktadır. Tez çalışmaası hakkında literatürde bulunan çalışmaları aşağıdaki başlıklar altında sıralayabiliriz:

i Anklaşman Sistemlerinin Model Tabanlı Doğrulanması

ii Anklaşman Sistemlerinin Sınanması

iii PLC sistemlerin doğrulanması

1.2.1 Anklaşman sistemlerinin model tabanlı doğrulanması

Birçok araştırmada Anklaşman Sistemlerinin sınanması yoğun işlem gerektiren SAT (Boolean Satisfiability) yaklaşımı ile yapıldığı görülmüştür. Sınama işlemi sırasında, öncelikle güvenlik unsurları olan; raydan çıkma, çarpışma gibi hayati durumları ve bununla birlikte kurulumun her bir trenin hareket edebileceğini de incelenmiştir.

Sınama problemini SAT problemi olarak modelleyen çalışmalar mevcuttur [1] [2] [3] [4]. Sınanması istenen durumlar formül olarak tanımlanıp, gereksinimler sistemin girdileri ve çıktıları üzerinde tanımlanmıştır (Örneğin X girdisi doğruysa Y çıktısı yanlış olmalı). Model sınayıcılar [5] ya da SAT problem çözücüler [6] sınanacak durumları sağlanıp sağlanmadığını hesaplamalar neticesinde gösterir. Bu yöntem ile bir anklaşman sisteminin çoğu durumu kontrol edildiği iddia edilse de kapsanan durum uzayı tüm tehlikeli durumların testini içermemektedir.

1.2.2 Anklaşman sistemlerinin model tabanlı test edilmesi

Otomatik model tabanlı sınama ve sınama durumu üreteci için geliştirilmiş bir takım araç literatürde mevcuttur [7] [8] [9] [10]. Bu araçlardan yalnızca JTorX isimli aracın literatürde anklaşman sistemlerinin sınanmasında kullanılmasına yönelik bir çalışma mevcuttur [11]. Öte yandan anklaşman sistemlerinin otomatik sınanması konusunda yapılan iki çalışma da simülatör tabanlı çalışmalardır [12] [13] ve modellenen sistemin yazılım tabanlı simüle edilerek, elle veya otomatik olarak türetilmiş durumların sınanması gerçekleştirilmektedir. Bu çalışmalardan ilki bu tez danışmanının da içinde bulunduğu UDSP projesi kapsamında olası durumların bir kısmını kapsayan otomatik senaryo üretimi ile çalışan bir simülasyon aracıdır [12] . Diğer çalışmada ise test durumları yalnızca manuel olarak üretilmektedir [13] . Yukarıda sözü edilen çalışmalar sınama durumlarının üretilmesinde durum indirgeme yöntemlerini kullanmakta ve hazırlanan modeller için garanti sonuç üretememektedirler.

1.2.3 PLC sistemlerinin doğrulanması

PLC yazılımlarının sınanması için, ilgili literatürde yapılan birkaç çalışmada NuSMV sembolik model sınayıcı kullanılmıştır [14] [15]. Bu çalışmalardan bir tanesinde

[15] gerçek bir saha ekipmanı üzerinde sınama gerçekleştirmiştir. Ayrıca PLC yazılımlarının UPPAAL modellerine dönüşümü ile ilgili bir çalışma da literatürde mevcuttur [16] fakat bu çalışma anklaşman sistemlerine uygulanmamıştır. PLC sistemlerinin model sınaması ile ilgili literatürde az çalışma bulunmasının nedeni PLC yazılımlarının fazla sayıda durum içermesi ve model sınamada durum sayısında patlama yaratmasıdır.

2. ANKLAŞMAN SİSTEMİ

2.1 Anklaşman Sistemi Elemanları

Tezçalışmasında model sınama işleminin üzerinde gerçekleştirileceği demiryolu anklaşman sistemi bileşenleri aşağıda sıralanmaktadır.

2.1.1 Sinyal

İşlevsel bir istasyon bölgesinde trenlerin ilerleyişini düzenlemekte kullanılan en önemli parçalar sinyallerdir. Bu sinyal tipleri anklaşman sisteminin verdiği kararlar doğrultusunda durumlarını değiştirerek demiryolunda seyahat eden trenlerin akışını düzenlemektedirler. Ülkemizde demir yollarında, cüce/yüksek ikili/üçlü/dörtlü gibi birkaç farklı tip sinyal kullanılmaktadır.

2.1.2 Makas

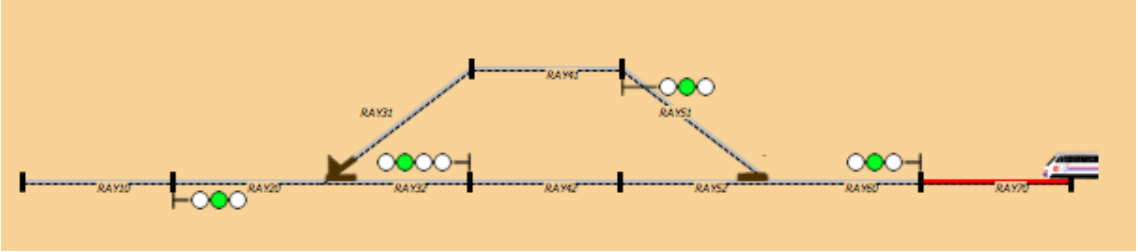
Makaslar raylar arasına yerleştirilerek trenlerin hat değiştirmesine olanak vermekte ve diğer saha elemanlarıyla koordine çalışarak hat değiştirme işleminin güvenli bir biçimde yapılmasını sağlamaktadır. Örneğin bir makas elemanı hat değiştirmeye olanak verecek şekilde durum değiştirirken aynı zamanda o elemandan önce gelen sinyal de "yavaşla" anlamına gelecek biçimde yanmalıdır.

2.1.3 Hemzemin geçit

Hemzemin geçitler tren hatlarının karayolları ile keşiştiği noktalarda bulunmaktadır. Hemzemin geçitlerin kontrolü ile ilgili en önemli özelliklerden biri geçitlerin kapanıp açılması sırasındaki zaman kısıtlarının diğer elemanlara oranla daha farklı olmasıdır.

2.1.4 Ray devresi

Ray devreleri belirli ray blokları üzerinde bulunup trenlerin rayın hangi noktasında bulunduğu konusunda bilgi vermekte kullanılmaktadır. Belirli bir ray devresinin



Şekil 2.1: Örnek demiryolu saha modeli.

meşgul durumda olması ilgili tren bloğunda bir tren bulunduğu bilgisini vermektedir. Bu bilgiler doğrultusunda anklaşman sistemi kararlar vererek yukarıda sözü edilen sistem elemanlarının durumlarını değiştirir.

2.1.5 Anklaşman yazılımı

Anklaşman yazılımının çalışması sırasında etkileştiği üç ana unsur vardır:

i Ray devrelerinden gelen tren konum bilgileri

ii Trenlerin sırasıyla hangi ray bokları üzerinde izleyeceğini bildiren tren rota bilgisi

iii Trenlerin güvenli bir biçimde ilerlemesini için durum atamalarını yaptığı ve geri bildirim aldığı saha elemanları(sinyal, makas, hemzemin geçit)

2.2 Anklaşman Sisteminin Model Sınaması

Yukarıda anlatılan özellikler doğrultusunda hazırlanan anklaşman sistemlerinin yazılımsal olarak sınanması için UDSP kapsamında basit bir saha modeli için üretilen simülatörün bir ekran görüntüsü aşağıda sunulmuştur. Yukarıda sözü edilen(anklaşman sistemi hariç) tüm unsurların gerçek PLC yazılımları ile etkileşimli olarak temsili Şekil 2.1'de mevcuttur.

Örnek modelde tek bir trenin tek bir yönde izleyebileceği farklı güzergah sayısı 5'tir. Bir tren, her bir güzergahta 4 farklı saha elemanının etkisi altına girmekte ve her bir saha elemanının içinde bulunabileceği farklı durum sayısı 1 ile 7 arasında değişmektedir(ortalama 4 durum). Tren durumlarının her saha elemanını geçtiğinde değiştiğini varsayarsak bir tren yukarıdaki istasyona girişinden çıkışına kadar Denklem 2.1 'deki hesapla 340 tane durum sayısı ile karşı karşıyadır. Bu durumların sahaya giren birden fazla tren olması ve bu trenlerin birbirlerine göre farklı zamanlarda sahaya giriş yapmaları söz konusu olduğunda durum sayısı gittikçe kaba kuvvet(brute-force)

sınamaya olanak veren bir sistemin yeteneklerini aşmaktadır.

$$\sum_{i=1}^{4} 4^i = 340 \tag{2.1}$$

Model sınama tekniklerinin kullanılmasını gerekli kılan tek neden sadece yukarıdaki durum uzayının soyutlanarak durum sayısının azaltılması değildir. Bunun yanında PLC çevrim sürelerinin uzunluğu nedeniyle gerçek PLC sistemlerin kullanıldığı testlerde en basit senaryoların bile sınanmasının dakikalar mertebesine ulaşabilmesidir. UDSP projesi kapsamında geliştirilen otomatik senaryo test eden yazılım tabanlı simülatör ile PLC'Ierin çevrim süreleri mümkün olduğunca hızlandırılarak yeterli test durumu günler mertebesinde yapılan testlerle gerçekleştirilebilmiştir. Öte yandan PLC çevriminin kavramsal olarak saat vuruşu kısalığında tutulabileceği gerçek zamanlı model sınama yöntemleri ile durum uzayı çok daha etkin biçimde sınanabilecektir.

Model sınama teknikleri yukarıda anlatılan PLC'Ierin aktif olarak kullanıldığı testler sonucu ortaya çıkan zaman kısıtlarının ortadan kaldırılmasında ve böylece daha büyük bir test kümesinin daha sistematik yollarla sınanarak garantili sonuçların elde edilmesinde kullanılacaktır. Model sınamanın bu bağlamda en büyük yararı modellerin oluşturulması/dönüştürülmesi sırasında bir takım soyutlamaların yapılarak sistem karmaşıklığının azaltılması ve bu şekilde sınanmak istenilen özelliğe odaklanılarak doğrulamanın gerçekleştirilebilmesidir. Örneğin doğrulanan sistemin gerçek zamanlı özellikleri yerine iş sıralama özelliğinin sınanması amaçlandığında sistem modelindeki zamanlar birim zamana çekilerek(ya da duruma göre tamamen göz ardı edilerek) durum uzayı küçültülebilir ya da arzulanan bir saha elemanının çalışmasına odaklanılmak isteniyorsa diğer saha elemanları detaysız ve en basit biçimde modellenerek durum uzayı yine küçültülebilir. Bu duruma örnek olarak makas elemanının sadece "AÇIK/KAPALI" durumları arasında gidip gelmesi ya da bu durumlar arasında belirli ara durumlara ve süre kısıtlarına bağlı modellenmesi sözü edilen detaylandırılma işlemlerindendir.

3. MODEL TABANLI SINAMA

3.1 Model Sınama

Model sınama biçimsel bir modelleme yöntemiyle modellenmiş bir sistemin, yine biçimsel belirtim teknikleri kullanılarak temsil edilen özelliklere sahip olup olmadığını sistematik yollarla kontrol etme işlemidir. Model sınamanın en önemli üç özelliği aşağıdaki gibi sıralanabilir:

i Sistem modelinin olası tüm durumlarını kontrol etmesi. Bu nedenle sistem modeli genellikle durum geçiş sistemleri olarak modellenir

ii Kolaylıkla otomatikleştirilebilir olması; günümüzdeki model sınama araçları yazılımsal olarak durum uzayını kontrol eder

iii Modelin desteklemediği bir belirtime ilişkin karşıt örnek sunabilmesi. Olası tüm durumlar kontrol edilirken belirtilen bir özelliği desteklemeyen duruma kadar olan durum geçişleri kullanıcıya geri bildirim amacıyla sunulur.

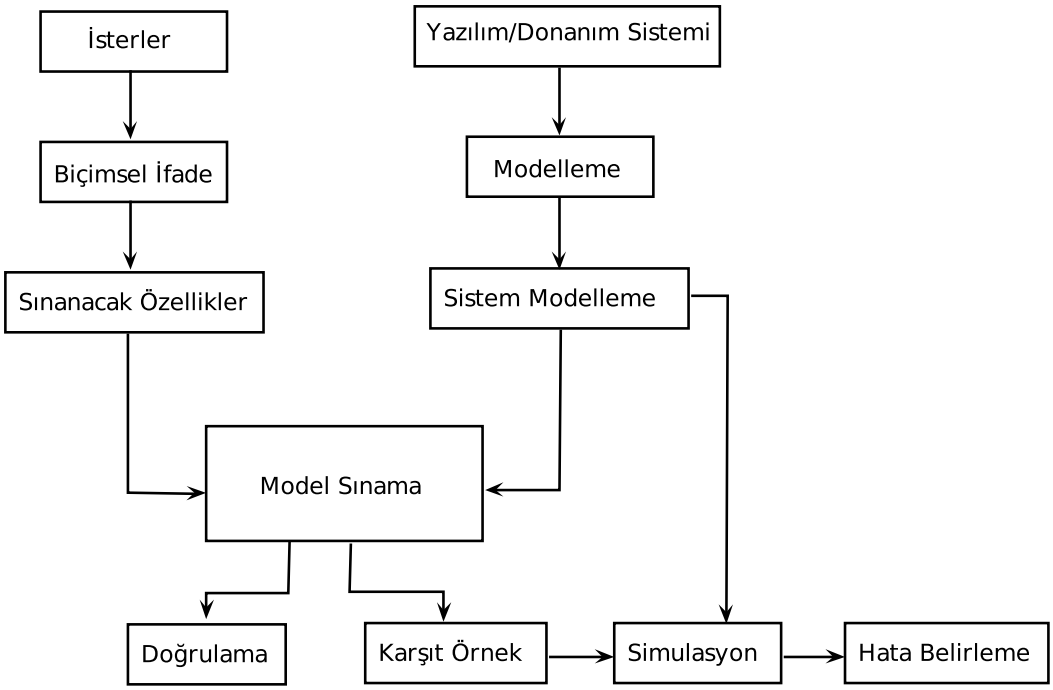
Yapılan tez çalışmasında gerçek zamanlı model sınamaya olanak veren UPPAAL kullanılacaktır. Bu aracın seçilmesinin nedenleri arasında:

i Açık kaynak kodlu versiyonlarının bulunması

ii Aracın model sınama konusunda en bilinen araçlardan olması

iii UPPAAL'ın gerçek zamanlı model sınamaya imkan vermesi olarak görülebilir

Şekil 3.1'de görülen çizimde, model sınama işlemi şematik olarak özetlenmiştir. Sol üst köşede sınanacak sistemden istenen özellikler, metin tabanlı ya da biçimsel belirtimlerden model sınama aracının desteklediği (Genellikle zaman belirtimi destekleyen LTL ve CTL gibi diller) biçimsel ifadelere dönüştürülmektedir. Sağ üstte görülen işlem ise sınanacak sistemin durum tabanlı modelinin(Zamanlı/zamansız



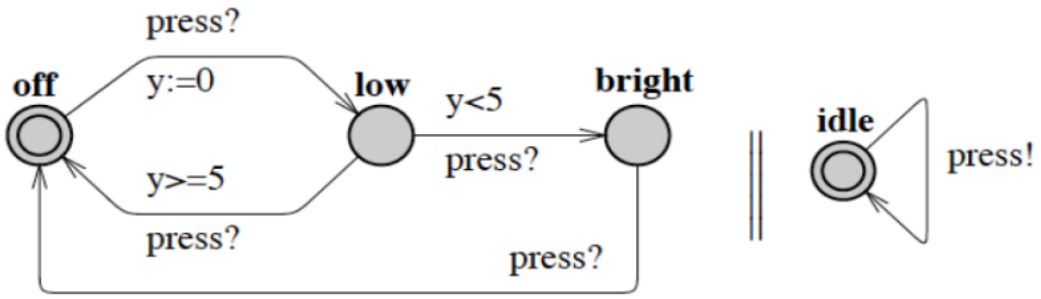
Şekil 3.1: Model sınama işlemi.

otomatlar) oluşturulması işlemidir. Bu modelleme sırasında yukarıda sözü edilen soyutlama işlemi de uygulanmaktadır. Model sınamanın sonucu olarak ya belirtilen model belirtilen özellikler için doğrulanır ya da belirtilen özelliğin desteklenmediği duruma yol açan durum geçişlerinin bulunduğu bir karşıt örnek simüle edilerek hatanın belirlenmesi sağlanır.

3.2 Gerçek Zamanlı Model Sınama

3.2.1 UPPAAL

Birçok model sınama aracı yukarıda görülen örneğe benzer biçimde gerçekleşen durum değişimlerinin sırasını doğrulayabilen göreli zamanlı belirtimi desteklerken gerçek zamanlı model sınamaya olanak veren model sınama araçları da bulunmaktadır. Bunların arasında sunduğu grafik arayüz desteği ile öne çıkan en yeni model sınama araçlarından biri UPPAAL'dır. UPPAAL modelleme dili olarak zamanlı otomatları kullanır ve doğrulanacak özelliklerde ve modellerde zaman notasyonunun kullanılmasına izin verir. UPPAAL ile modellenmiş basit bir lamba sisteminin modeli Şekil 3.2'de görülebilir. Şekilde y isimli değişken saat değişkeni olarak kullanılmakta olup model sınama çalışırken y değerinin her adımda artan değerler alabildiğini varsayarak durum uzayını oluşturur.



Şekil 3.2: UPPAAL ile model sınama örneği [17].

Zamanlı otomatlar ve UPPAAL kullanılarak modellenen sistemlerin özelliklerinin belirtilmesinde de zaman notasyonunu destekleyen diller kullanılmaktadır. Örneğin aşağıdaki belirtimde lambanın parlak(bright) duruma geçmeden önce loş(low) modunda 5 birim zamandan fazla kalamayacağı belirtilmiştir.

Çalışmada yukarıda belirtilen model sınama teknikleri ile pilot saha üzerinde bulunan elemanlar modellenerek anklaşman sistemi sınanmıştır. PLC'de koşan saha kontrol yazılımlarının kullanılacak durum tabanlı sistemlerin varyantları(otomatlar, petri ağları) kullanılarak üretilmiş olması nedeniyle ilk etapta PLC yazılımlarının model dönüşümleri ile kullanılacak araçların modelleme dilleri ile belirtilmiş modellere dönüşümü otomatik olarak gerçekleştirilecektir.

Daha sonra model sınamanın destekleyebileceği durum sayısında uygun olarak pilot bölge üzerinde mümkün olan en büyük durum uzayları farklı özelliklere odaklanılarak oluşturulacak ve model sınama tekrarlanacaktır. Tez çalışması çıktısı olarak sistemin ölümcül kilitlenme, kaza durumları ya da uyumsuz-çelişkili saha elemanı çalışmaları gibi durumlardan yoksunluğunun doğrulanması hedeflenmektedir.

Durum uzayı patlamasını engellemek amacıyla, Petri Ağlarının, içerisinde zamanı barındıran çeşitli uzantıları sunulmuştur. TPN(Time Petri Nets) [18] ve TAPN(Timed-Arc Petri Nets) [19] bunlara örnektir. Bu modellemelerden TAPN yaklaşımı kullanılarak TAPAAL isimli açık kaynak kodlu bir araç geliştirilmiştir [20].

TAPAAL aracında, jetonlara bir yaş değeri ve geçişler arasındaki yaylar üzerinde zaman aralıkları eklenmiştir. Bu zaman aralıkları ile jetonlar üzerindeki yaş değerleri kullanılarak durumlar arası geçişler gerçekleşir. TAPAAL aracı UPPAAL aracı ile aynı doğrulayıcı motoru kullanır. TAPAAL, TAPN modellerinin modelleme, benzetme ve doğrulamalarında kullanılabilecek ilk araçtır [21].

4. PLC PROGRAMLAMA TANIMLARI VE SOYUTLAMALARI

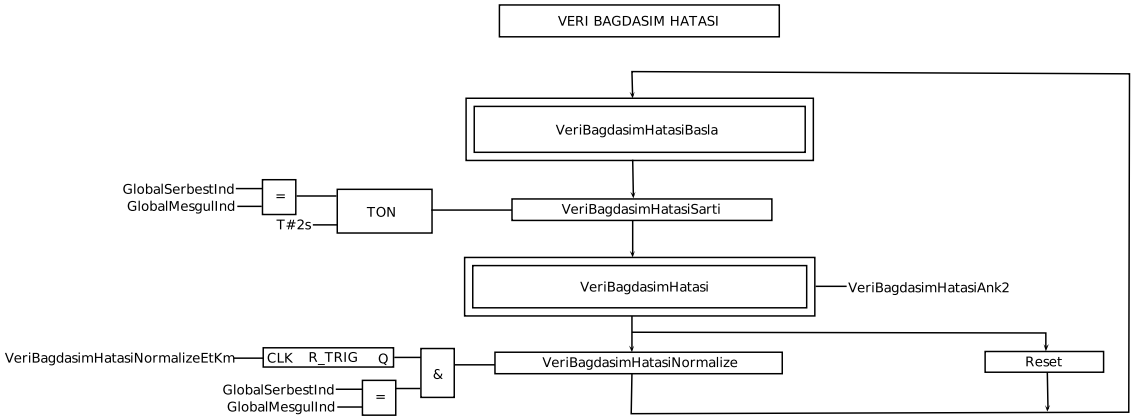
4.1 PLC Programlama Tanımları

PLC'ler çok yüksek miktarda I/O işlemi yapabilme yeteneğine sahip özelleştirilmiş bilgisayarlar olarak görülebilir. Bu işlemleri, gerçek zaman kısıtları altında gerçekleştirirler. Bilgisayar olması sebebiyle, PLC'lerin istenen amaçlar için programlanması gerekmektedir. Çok sayıda PLC üretici olması sebebiyle, endüstride PLC programlama konusunda IEC 61131 isimli bir standart getirilmiştir. Bu standart, PLC programlamada kullanılan bileşenleri (veri tipleri, değişkenler gibi) tanımlamıştır. PLC programlama dilleri genel olarak metin ve yada grafik tabanlıdır. Bu diller başlıca, IL(Instruction List), ST(Structured Text), FBD(Function Block Diagrams) ve LD(Ladder Diagrams).

PLC programlama, yoğun olarak bit seviyesinde yapılan işlemlerden dolayı geleneksel programlama dillerine göre daha alt seviyede çalışır. Bu yüzden PLC programlarında hata ayıklamak, test ve doğrulama yapmak oldukça güçtür. Daha önemlisi, PLC'ler en ufak hatanın tehlikelere yol açtığı gerçek zamanlı kritik işlerde kullanılmaktadır. Bu bağlamda PLC sisteminin doğruluğunun sınanması oldukça önemlidir.

PLC yazılımlarının doğrulanmasında biçimsel metotlar, 2 sebepten dolayı çok önemli yere sahiptir. Bunlardan birincisi, biçimsel metotların matematiksel güçlüğü, yazılım/modelin belirli kabuller altında doğruluğu ispatlanabilir. İkincisi ise, biçimsel metotları hedef sistem üzerinde farklı seviyede soyutlamalar yaparak uygulanabilmesidir. Böylece yazılım tasarımın ilk aşamalarında doğrulama yapmak mümkündür. Biçimsel metotların bu özelliği, çok ciddi hataları tasarımın ilk aşamalarında tespit edebilmesi sebebiyle oldukça önemlidir.

Model sınama tekniklerini PLC yazılımlarına uygulamak için, PLC yazılım modelini biçimsel dil modeline çevirmemiz gerekmektedir. Bu bağlamda bu tez çalışmasında PLC yazılım modelleri incelendi ve FBD dilinde tasarlanmış bir anklaşman sisteminin UPPAAL doğrulama dilinde doğrulama yapılacak şekilde model doğrulama diline



Şekil 4.1: Veri bağdaşım hatası FBD programı.

çevrimi yapılmıştır. Bunun için FBD programını ara model olarak Petri ağlarının bir çeşidi olan PNLF(Petri Net Linear Form) formatına çevirerek otomata çevirdik. Bu otomat, Java geliştirme dili kullanarak yazılan bir program ile UPPAAL model doğrulama diline çevrildi.

4.2 Fonksiyon Blok Diagramları

Fonksiyon Blokları, PLC programları için entegre devre eşdeğerliliği şeklinde ifade edilir. PLC tarafından sağlanan işlem kabiliyetlerini kullanarak istenen fonksiyonları yerine getirir. İyi şekilde belirtilmiş giriş/çıkış tanımlarına sahip olmasından dolayı, fonksiyon blokları PLC programcısı tarafından karakutu olarak da kullanılabilir.

FBD'ler PLC programı içesinde bilgilerin (işaret ve sinyallerin) bloklar arasındaki akışını gösterir. Örneğin Şekil 4.1'de, FBD farklı seviyelerde soyutlama yaparak birden fazla fonksiyon bloğunu içermektedir. Sistem karmaşıklığını soyutlamalar yaparak indirgemesinden dolayı, FBD'ler model sınamada oldukça popülerdir. Bu soyutlamalar sayesinde model sınama aşamasındaki durum uzayı azaltılmış olur.

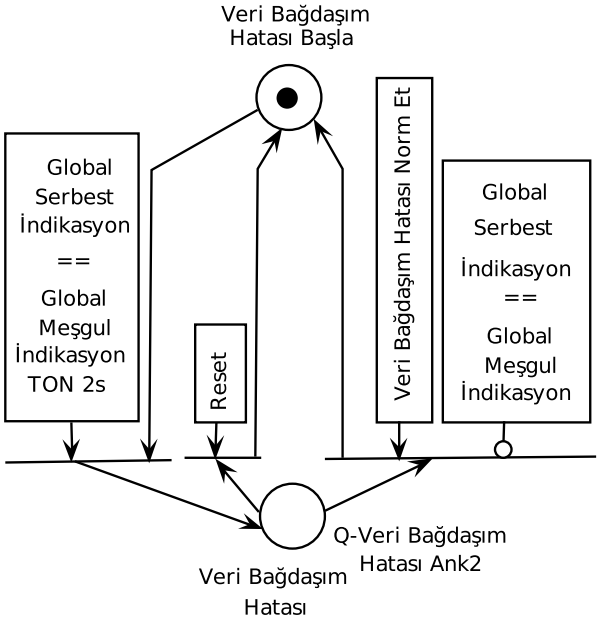
Şekil 4.1 de görülen FBD programında, anklaşman yazılımının "Veri Bağdaşım Hatası" ile ilgili kısmı görülmektedir. Kutu içerisinde belirtilen kısımlar fonksiyon blokları olup bu kısımlar soyutlanabilir. Programda, GlobalSerbestInd ve GlobalMesgulInd değişkenleri eşitlik testinden geçerek, bir sinyal olarak TON (Timer on Delay) devresine giriş olarak gelmiştir, 2 sn boyunca bu eşitlik sağlanması durumunda VeriBagdasimHatasiSarti sağlanmış olup, VeriBagdasimHatasi fonksiyon bloğu aktif olacaktır. Bu durumu diğer FBD program bileşenlerine iletmek için, VeriBagdasimHatasiAnk2 ismindeki bayrağı matıksal 1 değerine kurmaktadır. Benzer

şekilde VeriBagdasim bloğu aktif iken Reset değeri mantıksal 1 yapıldığında, akış başlangıç fonksiyon bloğu olan VeriBagdasimHatasiBasla'ya geçecektir.

4.3 Petri Ağları

Petri ağları, PLC programlarını biçimsel modellerde ifade etmek için yaygın olarak kullanılmaktadır. Bir Petri ağı, durumlardan, geçişlerden ve yaylardan oluşmaktadır. Durumlar üzerinde birden fazla jeton bulunabilir. Bu jetonlar petri ağlarında eşzamanlı çalışma olanağı sağlamaktadır. Jetonun bir durumdan diğerine geçmesi, bu iki durum arasında bulunan geçişler ve yaylar tarafından kısıtlanır. 2 ya da daha fazla durum birbirine 1 geçiş ve birden fazla yay ile bağlanabilir. 2 durum birbirine 1 adımda bağlanabilmesi için en az 1 geçiş ve 1 çift yay olması gerekir. Şekil4.2 deki bir programın Petri ağı modeli görülmektedir. Jetonlar, durumlar içerisindeki siyah noktalar olarak temsil edilmektedir ve geçişler vasıtasıyla durumlar arasında eş zamanlı çalışmayı temsil eder.

Petri ağları PLC program modellemede ve model sınama yöntemlerinde kullanılır. Bunun başlıca sebebi olarak, petri ağları diğer biçimsel modelleme yaklaşımlarından daha kolay bir şekilde PLC programlarına çevrilebilir. Ayrıca petri ağları güçlü bir araç ve analiz desteği olarak model sınama alanında kullanılır.PLC programlar modellenirken, Petri ağların, Sinyal Çevirmeli Petri Ağları, Renklendirilmiş Petri Ağları gibi çeşitli versiyonları kullanılır.



Şekil 4.2: Veri bağdaşım hatası petri ağı modeli.

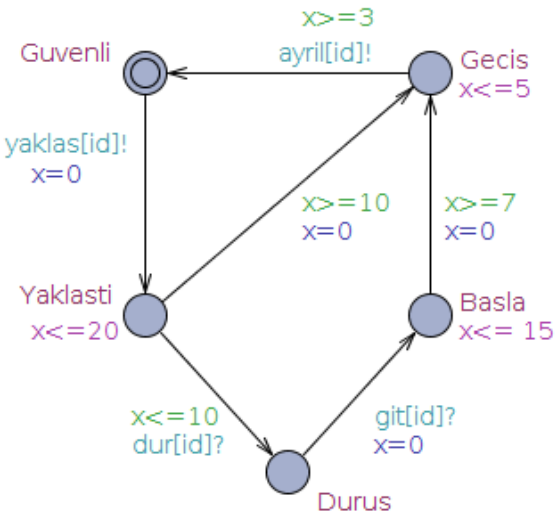
Şekil 4.1 de görülen FBD programına karşılık gelen Petri Ağı Şekil 4.2 de görülmektedir. FBD modelindeki fonksiyon blokları Petri ağında bu kısımlar durum olarak dönüştürülüp soyutlanmaktadır. FBD programındaki fonksiyon blokları arasındaki şartlar Petri ağında geçiş olarak ve bu şartların sağlanmasında kullanılan ifadeler ise sinyal olarak soyutlanmıştır.

4.4 Zamanlı Otomatlar

Kritik sistemler için genellikle gerçek zaman koşullarını göz önünde bulundurmak gerekmektedir. Belirli bir durum gerçekleşmeden belli bir süre önce ya da sonrasında sistemin başka duruma geçmesi gerekebilir (örneğin, sistemde oluşan kritik bir durumda, alarmın 5 saniye içinde aktif edilmesi gibi). Gerçek zamanlı sistemlerin davranışlarının modellenebilmesi için [22] zamanlı otomat modeli ortaya çıkmıştır. Bu modelleme yaklaşımında, gerçek zamanlı olma özelliğini klasik otomatlara "saat" bilgisi eklenerek elde edilmiştir. Sistemdeki saat bilgisi, bir değişken olarak gösterilip zamanla birlikte senkron bir şekilde artar. Durumlar arası geçişlerdeki şartlarda, saat bilgisi kullanılarak geçişlerin hangi zamanda ya da zaman aralığında gerçeklenmesini belirtebilir bu geçişlere bağlı güncelleme işlemlerini gerçekleştirebiliriz.

Zamanlı Otomatların en büyük dezavantajlarından birisi durum uzayı patlamasına meyilli olmasıdır. Sistemde eklenen herbir saat değişkeni, durum uzayını büyütmektedir. Büyük sistem modellerinde, durum uzayı patlamasından kaçınılması için, mümkün mertebede az sayıda sistem saati kullunılması amaçlanmalıdır.

Şekil 4.3'de birden fazla trenin ortak bir geçite erişimini kontrol eden demiryolu kontrol sisteminin UPPAAL'da modellenim zamanlı otomatı görülmektedir. Örnekte geçit aynı anda sadece bir trenin kullanabileceği kritik bir ortak kaynaktır. Bu bağlamda sistem modellenirken, birden fazla trenin varlığı gözönünde bulundurulmuştur. Modellemede, bir tren doğrudan "Duruş" konumuna anında geçemez ve "Durus" konumundan tekrar harekete geçmesi için zaman geçmesi gerekiyor. Modele bir trenin geçişe yaklaşması için belirli zaman kısıtlamaları konulmuştur. Bir tren yaklaşırken, yaklas! sinyalini gönderip "Yaklasti" konumuna geçer bu konumda iken 10 zaman birimi içerisinde dur? sinyalini alırsa güvenli bir şekilde "Duruş" konumuna geçer. Tren "Yaklasti" konumundayken 10 zaman birimi içerisinde dur? sinyalini almadıysa eğer, 10 zaman birimi içerisinde geçişe ulaşıp "Gecis" konumuna geçecektir. "Duruş"



Şekil 4.3: Tren gecidi kontrolü zamanlı otomat modeli.

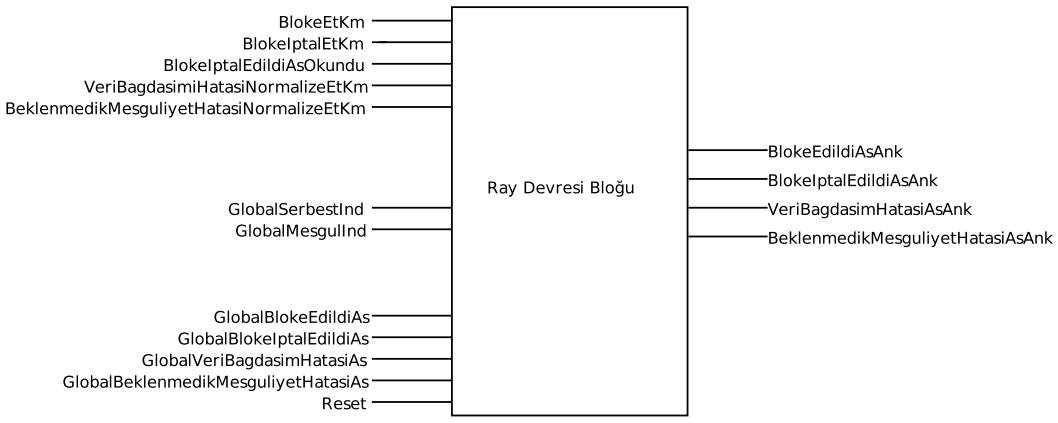
konumunda olan tren, ilerle? sinyalini aldıktan sonra harekete başlar. Geçidi kullanan tren ayrıldıktan sonra ayriliyor! sinyalini gönderir [17].

5. YAPILAN ÇALIŞMA

Bu tez çalışmasında önerilen sistem 4 aşamadan oluşmaktadır. Bunlar model sınaması yapılacak olan anklaşman sistemine ait FBD ile oluşturulmuş PLC Programlarının Petri ağlarına dönüşümü, Petri ağlarının PNLF dönüşümünün yapılması, oluşturulan PNLF ifadelerinin ayrıştırılması, UPPAAL modellerinin otomatik olarak oluşturulması ve son aşama olarak UPPAAL modellerinin sınamasının yapılmasıdır. Tez çalışmasında kullanılan demiryolu devreleri, 2009-2012 yılları arasında İTÜ ile TÜBİTAK BİLGEM ortaklığıyla gerçekleştirilmiş Ulusal Demiryolu Sinyalizasyon Projesi (UDSP) kapsamında İTÜ tarafından FBD ile tasarlanmıştır.

Petri ağları ile, yazılım tasarımlarında kullanılan ardışık çalışma, senkronizasyon, ve eş zamanlı çalışma gibi kavramlar kolaylıkla modellenebilmektedir. FBD de dahil olmak üzere birçok PLC programlama dili Petri ağına dönüştürülüp soyutlama yapılmaktadır [23] [24] [25] . Bu sebeple bu tez çalışmasında Petri ağ tercih edilmiştir.

Yapılan çalışma kapsamında, Anklaşman yazılımının Ray Devresi Bloğu'nun alt modulleri olan Beklenmedik Meşguliyet Hatası, Ray Bloke ve Veri Bağdaşım Hatası modellerinin sınaması yapılacaktır. Ray Devresi Bloğu'na ait FBD arayüzü Şekil 5.1'de gösterilmiştir. Arayüzde sol tarafta gösterilen sinyaller, bloğun giriş sinyalleri olup, sol tarafındakiler bloğun çıkış sinyalleridir.



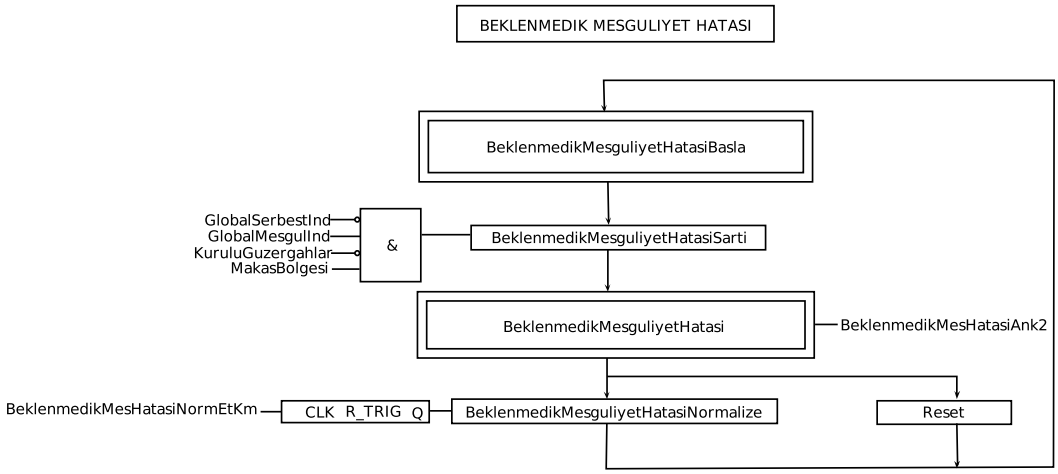
Şekil 5.1: Ray devresi bloğu.

Petri ağları grafiksel modelleme dili olduğu için, doğrudan model sınama diline çevrilememektedir. Bu bağlamda ara bir dönüşüm yapılması gerekmektedir. PNLF kullanılarak bu dönüşüm gerçeklenmiştir. Bölüm 5.3 de belirtilen gramer kuralları gözetilerek, tez çalışmasında ihtiyaç duyulan durumlarda bu gramer kuralları genişletilerek Petri Ağı- PNLF dönüşümü yapılmıştır.

5.1 FBD – Petri Ağı Dönüşümü

FBD ile yazılmış PLC programlarında, fonksiyon blokları Petri ağlarındaki karşılığı durum olarak, fonkisyon blokları arasındaki koşullar ise, durumlar arası geçişlerin şartları olarak dönüştürülmüştür. FBD grafiksel dilinin okunur olması sebebiyle bu dönüşüm manual olarak yapılabilir. FBD - Petri ağları dönüşümü UDSP kapsamında yapılmış olup, bu tez çalışmasında hazır olarak kullanılmıştır.

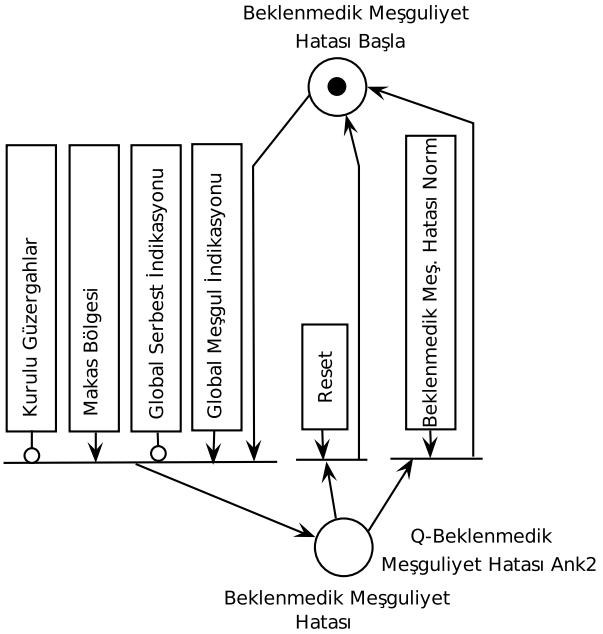
Anklaşman yazılımına ait Veri Bağdaşım Hatası fonksiyon bloğunun FBD programı ve petri ağı karşılığı bölüm 4.2 ve 4.3 de açıklanmıştır.



Şekil 5.2: Beklenmedik meşguliyet hatası FBD modeli.

Şekil 5.2 de, anklaşman yazılımının "Beklenmedik Meşguliyet Hatası" ile ilgili kısmı görülmektedir. Programda, KuruluGuzergahlar ve GlobalSerbestInd değerlerinin matıksal değil değerleri,MakasBolgesi ve GlobalMesgulInd değişkenleri mantıksal VE testinden geçerek, BeklenmedikMesguliyetHatasiSarti sağlanmış olup BeklenmedikMesguliyetHatasi fonksiyon bloğu aktif olacaktır. Petri ağı karşılığı olarak Beklenmedik Meşguliyet Hatası durumuna geçilecektir. Bu durumu diğer FBD program bileşenlerine iletmek için, BeklenmedikMeşguliyetHatasiAnk2 ismindeki bayrağı matıksal 1 değerine kurmaktadır. Benzer şekilde BeklenmedikMesguliyetHatası bloğu

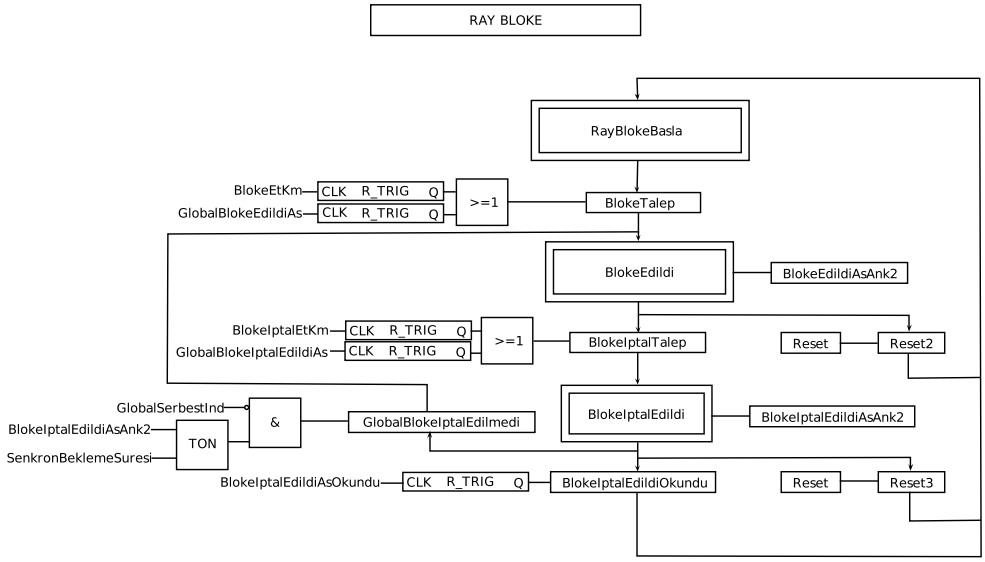
aktif iken Reset değeri mantıksal 1 yapıldığında, akış başlangıç fonksiyon bloğu olan BeklenmedikMesguliyetHatasiBasla'ya geçecektir. Beklenmedik Meşguliyet Hatası FBD programının Petri ağı dönüşümü Şekil 5.3'de gösterilmiştir.



Şekil 5.3: Beklenmedik meşguliyet hatası petri ağı.

Şekil 5.4 de, anklaşman yazılımının "Ray Bloke" ile ilgili kısmı görülmektedir. Programda, GlobalBlokeEdildiAs ve BlokEtKM değerlerinden herhangi biri mantıksal doğruysa, BlokeTalep sağlanmış olup BlokeEdildi fonksiyon bloğu aktif olacaktır. Petri ağı karşılığı olarak BlokeEdildi durumuna geçilecektir. Bu durumu diğer FBD program bileşenlerine iletmek için, BlokeEdildiAsAnk2 ismindeki bayrağı matıksal 1 değerine kurmaktadır.BlokeEdildi bloğu aktif iken Reset değeri mantıksal 1 yapıldığında, akış başlangıç fonksiyon bloğu olan RayBlokeBasla'ya geçecektir. BlokeTalebi fonkisyon bloğuna benzer şekilde, bloke iptali koşulları olan globalBlokeIptalEdildiAs ve BlokIptalEtKM değerlerinden herhangi biri mantıksal doğruysa, BlokeIptalTalep sağlanmış olup BlokeIptalEdildi fonksiyon bloğu aktif olacaktır.

Petri ağı karşılığı olarak Bloke İptal Edildi durumuna geçilecektir. Bu durumu diğer FBD program bileşenlerine iletmek için, BlokeIptalEdildiAsAnk2 ismindeki bayrağı matıksal 1 değerine kurmaktadır. Bu iptal talebi bekleme süresinden kısa süre içerisinde değerinin değişmesi durumunda GlobalBlokeIptal edilmedi koşulu sağlanamadığı için BlokeEdildi fonksiyon bloğu aktif olacaktır.BlokeIptalEdildiASOkundu sinyalinin değeri mantıksal doğru olduğunda BlokeIptalEdildiOkundu şartı sağlanıp,



Şekil 5.4: Ray bloke FBD modeli.

RayBlokeBasla bloğu aktif olacaktır. Ayrıca bu durumda Reset geldiğinde fonksiyon başlangıç durumunda geçecektir. Ray Bloke programının Petri ağı dönüşümü Şekil 5.5'de gösterilmiştir.

5.2 Petri Ağı – PNLF Dönüşümü

PNLF, Petri ağlarının XML ve grafiksel gösterimlerinde, kolaylık sağlaması ve kullanılabilir olması ortaya çıkmıştır [26]. Tez çalışmasında kullanılacak anklaşman devrelerinin sınama diline çevrilmesi için, modelin bileşenlerinin bilgisayar tarafından ayrıştırılabilir olması gerekir. Bu bağlamda grafik gösterimden işlenebilir metin tabanlı bir yapıya geçiş için Petri Ağı – PNLF dönüşümü yapılmıştır. Aşağıdaki kurallar gözetilerek Petri ağları elle PNLF formatında ifade edilmiştir:

i Bir ifade nokta ile bitmelidir

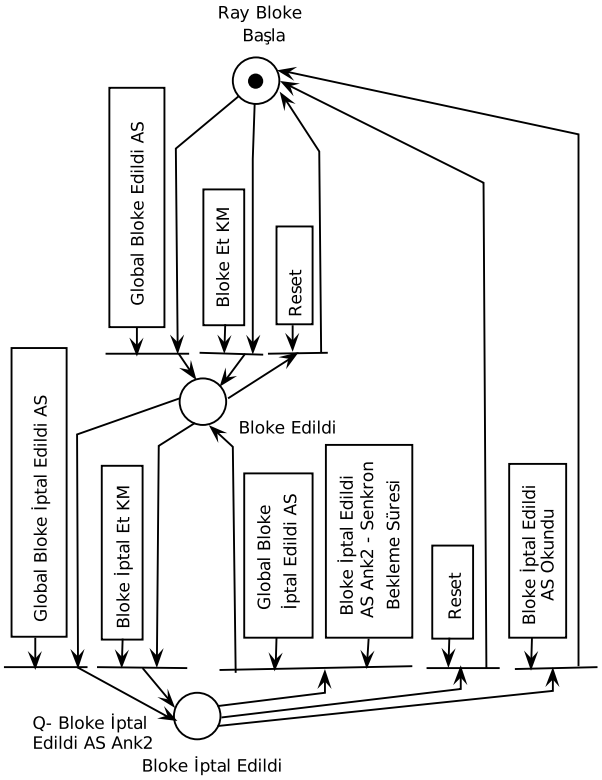
ii Birden fazla ifade noktalı virgül ile ayırılır

iii Durumlar paratezler içerisinde isimleri ile ifade edilir, örneğin (Durum1) gibi

iv Geçişler köşeli parantez içerisinde isimleri ile ifade edilir, örneğin [t1] gibi

v Yaylar yönlerine bağlı olarak ok işaretleri ile gösterilir, <– ve –> gibi

vi Jetonlar @ işareti ile belirtilir



Şekil 5.5: Ray bloke petri ağı.

vii Bir duruma gelen ya da durumdan çıkan yayların ve bunlarla ilişkili olan geçişlerin tümü, virgülle ayrılarak {} içerisine yazılabilir

viii Durum, geçiş ve yay değişkenleri, başında * işareti ile belirtilir

ix Satır yorumları başında // olarak yazılır, ya da /* */ içerisinde blok olarak yorum yazılabilir

x Geçişler arasındaki koşullar geçişlerin olduğu [] bloğu içerisinde "_" karakteri ile başlayarak eklenmelidir

xi Durum ile birlikte güncelleme yapılacak bir sinyal, durum tanımı ifadesine başında "+" karakteri olarak yazılır

Ray Devresi Bloğuna ait alt modullerinin PNLF ifadeleri aşağıdaki şekildedir.

```
Beklenmedik_Mesguliyet_Hatasi

(*BMHB @){
 ->[*t1 (~_kguz & _mbol & ~_gsi & _gmi)]->(*BMH +bmha2),
 <-[*t2 _reset]<-(*BMH),
 <-[*t3 _bmhNorm]<-(*BMH)
}.
```

```
Veri_Bagdasim_Hatasi

(*VBHB @){
 ->[*t1  (_gmi == _gsi)]->(*VBH +vbha2),
 <-[*t2 _reset]<-(*VBH),
 <-[*t3 (_vbhNorm & (_gmi != _gsi))]<-(*VBH)
}.

Ray_Bloke

(*RBB @){
 ->[*t1 _gbeAS]->(*BE +beasa2),
 ->[*t2 _beKM]->(*BE),
 <-[*t3 _reset]<-(*BE),
 <-[*t4 _reset]<-(*BIE +bieasa2),
 <-[*t5 _bieASO]<-(*BIE)
};
(*BE){
 ->[*t6 _gbieAS]->(*BIE),
 ->[*t7 _bieKM]->(*BIE),
 <-[*t8 (~ _gbieAS & _bieAS)]<-(*BIE)
}.
```

5.3 PNLF İfadelerinin Ayrıştırılması

PNLF ifadelerinin işlenebilmesi için, bir dönüştürücüye ihtiyaç vardır. Dönüşüm için bir gramer tasarlanıp, ANTLR(Another Tool For Language Recognition) isimli ayrıştırıcı ve Java programlama dili kullanılarak model üzerindeki bileşenler ayrıştırılmıştır. Ek 1 'de bulunan Ayrıştırma Gramer Kuralları'na ait ayrıştırma ağacı Şekil 5.6'de sunulmuştur. Ağaç üzerinde belirtilen "*" en az bir, "+" bir çok ve "?" sıfır ya da bir çoklayıcı ifadeleri yerine kullanılmıştır.

Ayrıştırma gramerine ait bileşenleri genel açıklamaları şu şekildedir.

5.3.1 pnlf

Ayrıştırılacak olan PNLF ifadesinin kök elemanıdır, PNLF ifadesi içerisinde bulunan durumlar, geçişler, jetonlar ve sinyaller bu kök elemanı altında toplanmıştır.

5.3.2 statementsWithinTheSameScopeForVariables

Durum çiftleri PNLF ifadesinde “;” karakteri ile ayrılırlar. Gramer ayrıştırma ağacında bu eleman yardımıyla ayrıştırılır. Durum çiftleri içerisinde bulunan sinyallerin ayrıştırılması amacıyla değişken tanımına benzer şekilde isimler tanımlanabilir.

5.3.3 placeStatement

Bu ayrıştırma elemanı ile bir durum ve bu durumun bağlı olduğu bütün geçiş ifadeleri sıralanmaktadır. Bu durumlardan çıkan ve bu durumlara giren geçişler durum ile geçiş ifadesi arasına konan yönlü oklarla belirtilmektedirler.

5.3.4 transition

Geçiş ifadeleri bu ayrıştırma elemanı ile temsil edilir. Köşeli parantezleri içinde tanımlanır ve birçok sinyal ile farklı petri ağları ile senkronize olması sağlanır.

5.3.5 place

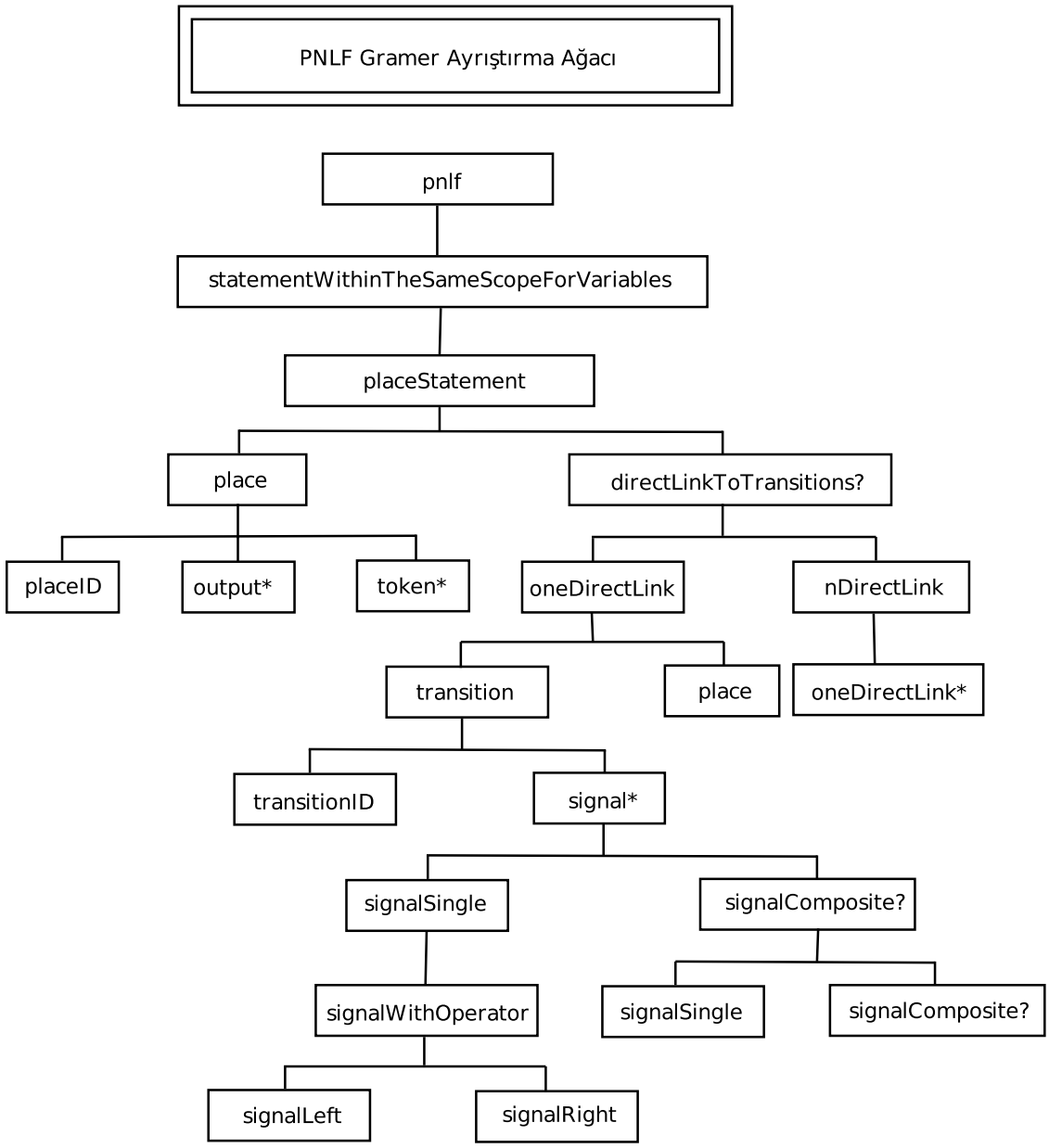
Durum ifadeleri bu ayrıştırma elemanı ile temsil edilir. Normal parantezler arasında tanımlanan bu elemanlar içerisinde ‘@’ işaretleri durumda bulunan jeton sayısını temsil etmekte kullanılmasıdır. Benzer şekilde “+” işaretleri durumdaki çıkış sinyalini temsil eder.

5.3.6 signal

Geçiş ifadeleri içerisinde belirtilen sinyaller senkronizasyon için kullanılırlar. PNLF gramerinde olmayan bu özellik, çalışmada ihtiyaç duyulduğu için eklenmiştir. Geçişteki bir sinyal mantıksal “NOT” operatorü gibi tekil olabileceği gibi mantıksal “AND” gibi bileşke bir sinyal ifadesi olarak da PNLF’e eklenebilir.

5.3.7 signalComposite

Bileşke sinyaller temel mantıksal operatörlerle (VE,VEYA, XOR.) bir araya getirilmiş sinyal ifadeleridir.



Şekil 5.6: PNLF gramer ayrıştırma ağacı.

5.3.8 signalSingle

Tekil sinyaller herhangi bir mantıksal operatör içermeyen tek bir sinyal içeren bir ifadelerdir.

5.3.9 signalWithOperator

Karşılaştırma içeren sinyali temsil eden ayrıştırma elemanı.

5.4 UPPAAL Modellerinin Otomatik Üretilmesi

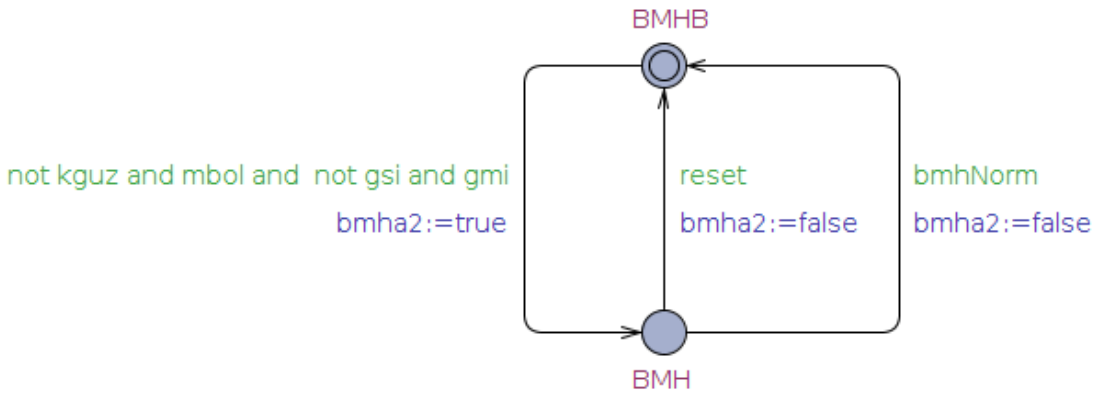
Uppaal modelleri grafik tabanlı modellerdir ve içerisinde zaman değişkenlerinin kullanılmasına olanak sağlarlar. UPPAAL'da modeller XML ile ifade edilirler. Moduler bir yapı geliştirmeye olanak sağlayan UPPAAL'da modüller "template" olarak adlandırılır. Bir modelde birden fazla modül olabilir. Bir template içerisinde sırasıyla, değişken tanımları, durum tanımlar ve geçiş tanımları yapılır. Bu tanımlar "system" hiyerarşisi altında üst seviye tanım olarak kullanılabilir.

Ayrıştırma grameri ile çözümlenen sinyaller UPPAAL modelinde, "declaration" hiyerarşisi altında boolean değişkenler olarak tanımlanıp, varsayılan değer olarak false değerleri atandı. Durum uzayının şişmesini bir nebze de olsa azaltmak için modül senkronizasyonunda ortak değişkenler kullanılmıştır.

Ayrıştırma grameri ile çözümlenen durumlar UPPAAL modelinde "location" hiyerarşisi altında tanımlanmıştır.UPPAAL'ın görsel olması nedeniyle, editör üzerinde bu durumların koordinatları varsayılan olarak sabit koordinat olarak oluşturuldu. Bu durum modelin sınanması açısından herhangi bir sorun teşkil etmemektedir.

Ayrıştırma grameri ile çözümlenen geçişler UPPAAL modelinde "transition" hiyerarşisi altında tanımlanmıştır. Geçişin başlangıç noktası olan durum ismi "source" çocuğu altında, hedef durum ismi ise "target" çocuğu altında belirtilmiştir. Geçiş senkronizasyonunda kullanılan tekil ve bileşke sinyaller ise "label" çocuğu altında "guard", atamalar ise "assignment" tipi ile belirtilmiştir.

Ayrıştırma grameri ile çözümlenen jetonlar UPPAAL modelinde "init ref" etiketi ile tanımlanmıştır. Petri ağlarında bulunan çoklu jeton desteği dönüşümde ele



Şekil 5.7: Beklenmedik meşguliyet hatası UPPAAL modeli.

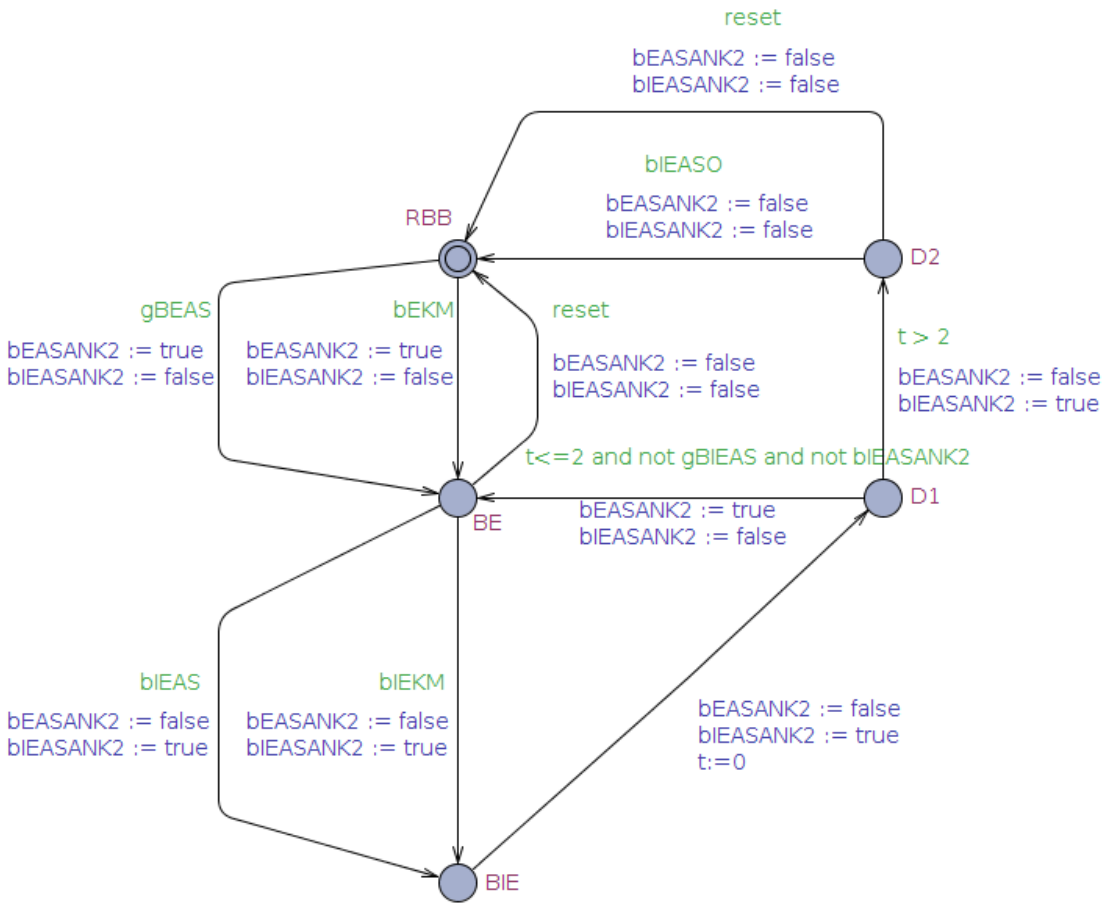
alınmamıştır. Elimizde bulunan anklaşman devrelerinde çoklu jetonun oldu bir FBD örneği de olmadığından çoklu jeton desteğini geliştirmeye de gerek duyulmamıştır.

Şekil 5.7 de Beklenmedik Meşguliyet Hatası devresine ait otomatik dönüşümü yapılmış olan UPPAAL modeli gösterilmiştir. Devrenin şekil 5.3'te gösterilen petri ağından da anlaşılacağı gibi Beklenmedik Meşguliyet Hatası durumunda Beklenmedik Meşguliyet Hatası Ank2 sinyalinin mantıksal 1'e çekilmesi gerekiyor. Bu bağlamda otomatik dönüşüm sırasında bu sinyalin değeri ilgili geçişinde mantıksal 1, diğer tüm geçişlerde ise matıksal 0 olarak atanmıştır.

TON tipi gecikme içeren devrelerin, otomatik üretilen UPPAAL modelleri üzerinde model sınamanın daha etkin bir şekilde yapılması için bir takım değişiklikler yapılmıştır. Bölüm 5.5'da bu değişiklikler detaylı olarak anlatılmıştır. Ek 2'de Ray devresine ait tüm modüllerin otomatik üretilen UPPAAL modellerinin XML çıktısına ulaşılabilir.

5.5 TON Tipi Gecikmelerin Modellenmesi

PLC program modüllerinde sıklıkla kullanılan Timer-Delay ON (TON) yapısı Petri ağlarında desteklenmediği için kullanılmamıştır. Otomatik üretilen UPPAAL modellerine TON durumları ek olarak gerçeklenmiştir. Literatürde TON'ların UPPAAL modellerine otomatik entegrasyonu gerçeklendiği çalışmalar mevcuttur [27] [28] [29]. Bu çalışmada ise modellenen yazılımlarda bulunan toplam TON sayısı ele alınabilir düzeyde olduğu için UPPAAL modellerine dönüşüm durum değişmezleri(invariant) ve geçiş kuralları düzenlenerek ele alınmıştır.



Şekil 5.8: Ray bloke devresine ait TON tipi gecikmeli UPPAAL modeli.

Model sınaması yapılacak olan PLC programında bulunan TON tipi, gecikmeli etkiler zamanlı olarak modellenmiştir Şekil 5.8'de Ray Bloke devresi içerisinde bulunan TON tipi gecikme yapısına çözüm teşkil eden örnek bir zamanlı otomat sunulmuştur. Ray Bloke devresinde Bloke İptal Edildi durumundayken, 2 zaman birimi içerisinde bloke iptal edildi AS Ank2 sinyalinin değeri mantıksal 0 olması durumunda, Bloke Edildi durumuna geçilecektir. Bu yapı UPPAAL'da bir saat ve 2 durum eklenerek gerçeklenmiştir. Bloke Iptal Edildi durumundayken nondeterminist olarak D1 durumuna geçilir ve saat değeri 0'a ayarlanır. 2 saniye içerisinde bloke iptal edildi AS Ank2 değeri korunmuşsa D2 konumuna geçecektir. Aksi durumunda ve global bloke iptal edildi as sinyalinin değeri mantıksal 0 olması durumunda BE durumuna geçilecektir. BIE ve D2 durumları birlikte petri ağındaki Bloke İptal Edildi durumuna karşılık gelmektedir.

Kullanılan TON modellemesi, her bir devre elemenanı için en az 1 yeni durum ve bir saat değişkeni eklenmesini gerektirmektedir. Bu sebeple mevcut durum uzayında

bir artış görülecektir. Çok sayıda TON içeren sistemlerin model sınaması yapılırken durum uzayı patlaması yaşanacaktır. Literatürde bu problemle karşılaşılan çalışmalar tespit edilmiştir. [30] Bu problemi aşmak için Promela ve SPIN model sınama dillerini kullanmayı tercih etmişlerdir. [31] Caesar/Aldebaran Geliştirme Paketini kullanarak durum uzayını azaltmıştır. [28] ise 4 farklı modul üretmiştir; "Kordinatör" PLC senkronizasyonunu modellemek için, "Program" PLC programı için, "Çevre" I/O birimleri için, "Kesme" zaman bazlı kesmeleri modellemek için. İlk 3 modul bir çok çalışmada kullanılmış olmasına rağmen kesme modulü bu çalışmaya özgüdür [32].

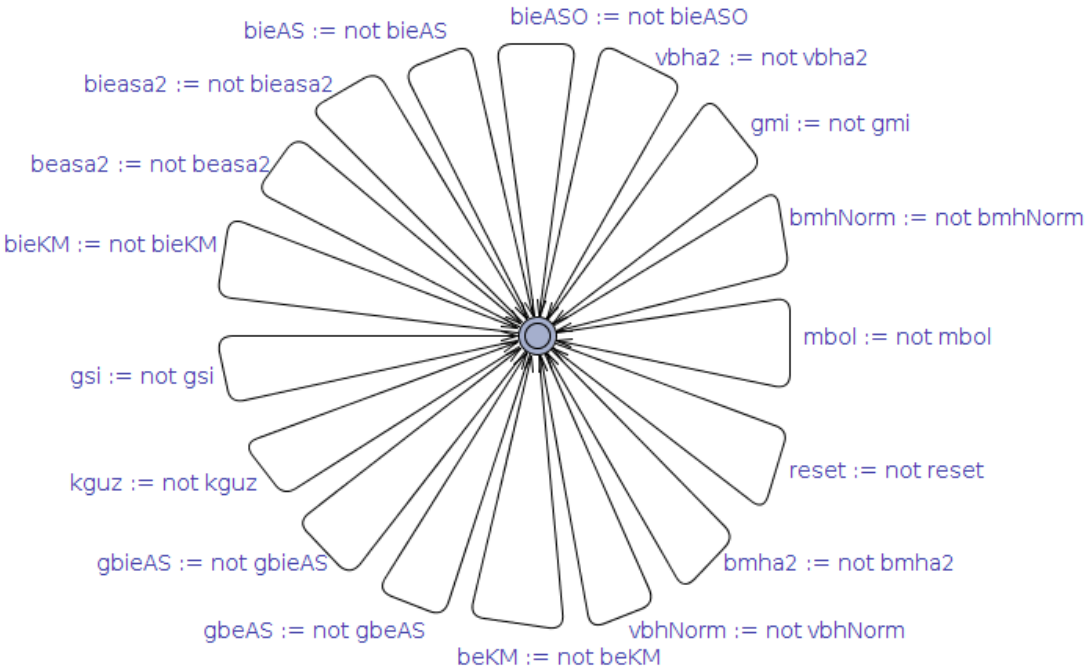
5.6 UPPAAL Modellerinin Sınanması

Tez çalışmasında her bir devre modeline ve sisteme ait ulaşılabilirlik, canlılık ve ölümcül kilitlenme özellikleri sınanmıştır. Bu sınamaların ilgili sorguları UPPAAL modellerindeki her durum elemanı için otomatik olarak üretilmiştir. Beklenmedik Mesguliyet Hatası modeline ait otomatik üretilen sorgular aşağıdaki gibidir.

```
A<> Beklenmedik_Mesguliyet_Hatasi.BMHB
E[] Beklenmedik_Mesguliyet_Hatasi.BMHB
E<> Beklenmedik_Mesguliyet_Hatasi.BMHB
A<> Beklenmedik_Mesguliyet_Hatasi.BMH
E[] Beklenmedik_Mesguliyet_Hatasi.BMH
E<> Beklenmedik_Mesguliyet_Hatasi.BMH
E<> deadlock
A[] not deadlock
```

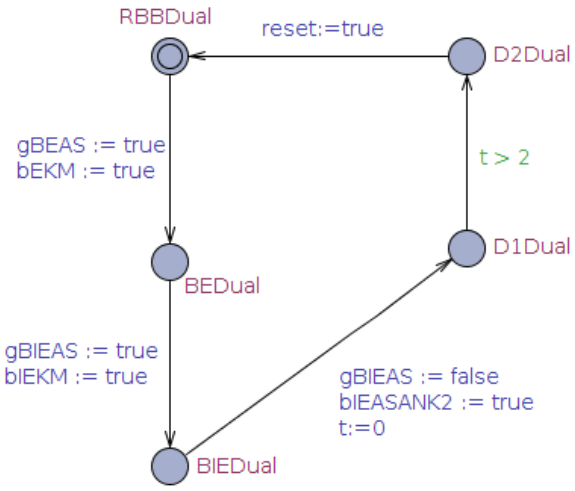
Ulaşılabilirlik sınamasında **E<> Beklenmedik_Mesguliyet_Hatasi.BMHB** sorgusu kullanılmıştır. Bu sorguda, nihayetinde en az bir kere bu duruma geçildiği sınanmıştır. Canlılık sınamasında ise **A<> Beklenmedik_Mesguliyet_Hatasi.BMHB** sorgusu kullanılmıştır. Bu sorguda, her zaman nihayetinde bu duruma geçildiği sınanmıştır. Sistemde ölümcül kilitlenme sınaması için 2 sorgu kullanılmıştır. **E<> deadlock** sorgusu ile sistemde ölümcül kilitlenme olduğunu sorgularken, **A[] not deadlock** sorgusu ile sistemde ölümcül kilitlenme olmadığını belirterek sınama yapılmaktadır. 2. sorguda ölümcül kilitlenme olması durumunda sınayıcı karşıt örnek vermektedir.

Otomatik üretilen modelde, sistemdeki tüm değişkenlerin varsayılan değerleri false olarak atanmıştır. Canlılık sınaması yapılırken tüm otomatların başlangıç durumundan ayrılmadığı tespit edildi ve sonrasında UPPAAL'ın model sınaması yaparken adillik(fairness) özelliğinin olmadığı tespit edildi. Bu bağlamda model sınama



Şekil 5.9: Ray devresine ait determinist olmayan değer belirliyici otomat.

işlemlerinde ek soyutlamalar gerekliliği ortaya çıkmıştır. Şekil 5.9'da gösterilen harici bir otomatın, diğer otomatlarda kullanılan tüm değişkenlerin değerini determinist olmayan bir şekilde değiştirmesiyle bu probleme çözüm aranmıştır. Fakat bu durumda da model sınama sırasında bir değişkenin hiç durmadan değer değiştirmesi durumunda sistemin canlı kilitlenme durumuna geçmesine sebep olmaktadır.



Şekil 5.10: Ray bloke devresine ait dual UPPAAL modeli.

Canlı kilitlenme durumundan kaçınmak için, herbir modelin duali tasarlanıp model sınama sırasında tüm durumlara geçildiğinden emin olunmuştur. Şekil 5.10'da ray bloke devresinin dual modeli görülmektedir. Bu model ile Şekil 5.8'de

gösterilen UPPAAL modelinin sınaması yapılırken adillik özelliği zorlama(brute force) biçimde gerçeklenmiş oldu. Bu devrenin model sınaması yapılırken tüm durumlara geçildiğinden, çıkış sinyallerinin değerleri de değiştirilmiş oldu. Bu yöntemin durum uzayı büyüklüğü açısından negatif etkisi oldu ve bu sebeple model sınama kısmi olarak yapıldı. Tüm anklaşman sisteminin kontrolü yapılamamıştır.

6. SONUÇ

Yapılan tez çalışmasında, UDSP kapsamında FBD PLC programlama dili ile hazırlanmış bir anklaşman yazılımının Petri ağı modellerini PNLF formatına çeviren bir gramer hazırlanıp, bu gramer vasıtasıyla petri ağı üzerinde bulunan bileşenler ANTLR Java kütüphanesi ile ayrıştırılmıştır. Daha sonra bu ayrıştırıcı kullanılarak bu PNLF ifadelerinin UPPAAL modeli karşılıkları ve model sınama sırasında kullanılan sorgular Java programlama dilinde yazılmış bir yazılımla otomatik olarak üretilmiştir. Üretilen UPPAAL modellerin model sınaması yapılmıştır.

PLC programlarında bulunan TON zamanlayıcılarından dolayı durum uzayı patlaması sebebiyle uygulanan soyutlama ve indirgemeler anlamlı sonuçlar alınmasına engel olmuştur. UPPAAL model sınama aracının adillik özelliği olmamasından dolayı, model sınama esnasında ek modellere ihtiyaç duyuldu. Eklenen bu modellerlerden bazıları anlamlı sonuç vermemişken, bazıları da durum uzayı patlamasına sebep olup sistemin kısmi olarak model sınamasına olanak sağlamıştır. Ray devresine ait 3 FBD yazılımının model sınama sonuçları aşağıdaki çizelgede gösterilmiştir.

Çizelge 6.1: Ray devresi FBD yazılımı model sınama sonuçları.

Model Adı	Nondeterminist Otomat	Dual Model
Beklenmedik Meşg. Hatası	Canlı Kilitlenme	Sınama Yapıldı
Veri Bağdaşım Hatası	Canlı Kilitlenme	Sınama Yapıldı
Ray Bloke	Canlı Kilitlenme	Durum Uzayı Patlaması

Tez çalışması için TAPAAL aracı da denenmiş olup, geçişlerde kullanılan mantıksal ifadeler kullanılamadığından ve değişkenlere değer atanamadığından kullanılmaktan vazgeçilmiştir.

Çalışmada yapılan ek modeller durum uzayı patlamasına sebep olmuştur. İleriye yönelik çalışmalarda bu durumunun engellenmesi için, UPPAAL dışında adillik özelliği olan ve gerçek zamanlı model sınamaya olanak sağlayan başka bir model sınayıcısı tercih edilebilir. Yapılan çalışmada, bu şekilde yeni bir model sınayıcı modeli üretmek kolaylıkla yapılabilmektedir. Ayrıca FBD PLC programı dili yerine,

IL ve LD gibi dillerde yazılmış PLC programlarının da model sınaması bu çalışmada önerilen şekilde yapılabilir.

KAYNAKLAR

[1] **Behrmann G., D.A. ve G., L.K.**, A Tutorial on Uppaal 4.0, http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf, updated November 28, 2006.

[2] **Groote, J., V.S. ve J., K.** (1994). The safety guaranteeing system at station hoorn-kersenboogerd, *Logic Group Preprint Series*.

[3] **Groote, J., V.S. ve J., K.** (1995). Safety guaranteeing system at station hoorn-kersenboogerd (extended abstract), 57–68.

[4] **W., F.** (1996). Safety criteria for the vital processor interlocking at hoorn-kersenboogerd, ***96***, 101–110.

[5] **A., B.** (1998). Case study: Formal verification of a computerized railway interlocking, *Formal Aspects of Computing*, ***10***, 338–360.

[6] **Cimatti A., Clarke E., G.F. ve M., R.** (1999). NuSMV: A new symbolic model verifier, *Computer Aided Verification*, 682–682.

[7] **Claessen K., Een N, S.M. ve N., S.** (2008). SAT-solving in practice, *Discrete Event Systems*, 61–67.

[8] **G.J., T. ve H., B.** (2003). Torx: Automated model-based testing, 31–43.

[9] **C., J. ve T., J.** (2005). TGV: theory, principles and algorithms, *International Journal on Software Tools for Technology Transfer (STTT)*, ***7***, 297–315.

[10] **A., H. ve K., N.** (2004). The AGEDIS tools for model based testing, ***29***, 129–132.

[11] **Veanes M., Campbell C., G.W.S.W.T.N. ve L., N.** (2008). Model-based testing of object oriented reactive systems with Spec Explorer, 39–76.

[12] **Sjitema M., Stoelinga M., B.A. ve L., M.** (2011). Experiences with formal engineering: Model-based specification, implementation and testing of a software bus at Neopost, 117–133.

[13] **Mutlu İ., Ovatman T., S.M. ve L., S.** (2011). A new test environment for PLC based interlocking systems, 686–690.

[14] **Calame J., Goga N., I.N. ve J., P.** (2006). Ttcn-3 testing of hoorn-kersenboogerd railway interlocking, 620–623.

[15] **Gourcuff V., S.O. ve J., F.** (2006). Efficient representation for formal verification of PLC programs, 182–187.

[16] **O., P. ve H., E.** (2010). Model checking PLC software written in function block diagram, in Software Testing, Verification and Validation (ICST), 439–448.

[17] **Soliman D., T.K. ve G., F.** (2012). Transformation of Function Block Diagrams to UPPAAL timed automata for the verification of safety applications, ***36***, 338–345.

[18] **P.M., M.** (1962). A Study of the Recoverability of Computing Systems, *Doktora Tezi*, Irvine,California.

[19] **Bolognesi T., L.F. ve S., T.** (1990). From Timed Petri Nets to Timed LOTOS, 395–408.

[20] **Jacobsen L., Jacobsen M., M.H.M. ve J., S.** (2010). A Framework for Relating Timed Transition Systems and Preserving TCTL Model Checking.

[21] **Byg J., J.K.Y. ve J., S.** (2010). TAPAAL: Editor, Simulator and Verifier of Timed-Arc Petri Nets.

[22] **R., A. ve L., D.D.** (1994). A theory of timed automata, ***126***, 183–235.

[23] **G., F. ve F., W.** (2006). A toolbox for the development of logic controllers using petri nets, 473–474.

[24] **I., G. ve M., A.** (2011). Model checking of control interpreted petri nets, 621–626.

[25] **I., G.** (2012). Control interpreted petri nets–model checking and synthesis.

[26] **Martin, P.A.** (2007). The Petri Net Linear Form (PNLF), *Doktora Tezi*.

[27] **Zoubek B., R.J.M. ve M., K.** Towards automatic verification of ladder logic programs.

[28] **Zhou M., He F., G.M. ve X., S.** (2009). Translation-based model checking for plc programs, ***1***, 553–562.

[29] **Mokadem H. B., Berard B., G.V.S.O.D. ve M., R.J.** (2010). Verification of a timed multitask system with uppaal, 921–932.

[30] **Mader A., Brinksma E., W.H. ve N., B.** (2001). Design of a plc control program for a batch plant vhs case study 1., 416–439.

[31] **H., W.** (1999). Compact timed automata for plc programs Tech. Rep. CSIR9925.

[32] **Ovatman T., Aral A., P.D. ve Ünver A. O** (2014). An Overview of Model Checking Practices on Verification of PLC Software, *Software & Systems Modelling*.

EKLER

EK 1 : PNLF Ayrıştırma Grameri
EK 2 : Ray Devresine Ait Otomatik Oluşturulmuş UPPAAL Modeli

EK 1

```
grammar PNLFCompiler;

options {
  language = Java;
}

@lexer::header {
  package tr.edu.itu.iLockCheck.parser.base;
}
@parser::header {
  package tr.edu.itu.iLockCheck.parser.base;

}


pnlf                                        :
    (statementsWithinTheSameScopeForVariables PERIOD) + ;
statementsWithinTheSameScopeForVariables  :
    placeStatement ( SEMICOLON placeStatement )* ;

placeStatement                              :
    place directLinksToTransitions? ;

transition                                  :
    LSQUARE transitionId signal* RSQUARE ;
transitionId                                :
    IDENTIFIEROFVARIABLE ;

place                                       :
    LPAR placeId output* token* RPAR ;
token                                       :
    ATSIGN ;
placeId                                     :
    IDENTIFIEROFVARIABLE ;
output   :
    OUTPUTSTR ;

signal                                      :
    signalSingle | LPAR signalComposite RPAR ;

signalComposite   :
    signalSingle (LOGOP signalComposite)?;

signalSingle   :
    LPAR? SIGNALNEG? (SIGNALSTR|signalWithOperator) RPAR?;
```

```
signalWithOperator   :
    LPAR signalLeft SIGOP signalRight RPAR;

signalLeft   :
    SIGNALNEG? SIGNALSTR;

signalRight          :
    (SIGNALNEG? SIGNALSTR)|POSITIVEINTEGER;

directLinksToTransitions                  :
    oneDirectLinkToTransition | nDirectLinksToTransitions ;

oneDirectLinkToTransition                 :
    (LEFTARR | RIGHTARR) (transition)| (LEFTARR | RIGHTARR)
    (transition) (LEFTARR | RIGHTARR) (place);

nDirectLinksToTransitions                 :
     LCURLY oneDirectLinkToTransition (COMMA
     oneDirectLinkToTransition)* RCURLY ;


WHITESPACE                                :
    ( '\n' | '\t'| '\r' | ' ' | ' ' )+  -> channel(HIDDEN) ;

COMMENT                                   :
    '//' ~( '\r' | '\n' )* -> channel(HIDDEN) ;

ML_COMMENT_STAR                           :
    '/*' .*? '*/'  -> channel(HIDDEN) ;

ML_COMMENT_HAT                            :
    '/^' .*? '^/'  -> channel(HIDDEN);

ANNOTATIONSTRING                          :
    '(^' .*? '^)' ;

POSITIVEINTEGER                           :
    (DIGIT)+ ;

SIGOP                                    :
    '==' | '!=' ;

LOGOP                                    :
   '&' | '|' | 'X' | '~X' ;

SIGNALNEG   :
    '~';

SIGNALSTR                                 :
    '_'(ALPHANUM)+ ;

OUTPUTSTR                                 :
```

```
    '+'(ALPHANUM)+ ;

fragment DIGIT                          :
    '0'..'9' ;

YES                                     :
    'yes->' | 'yes';

NO                                      :
    'no->' | 'no' ;

LEFTARR                                 :
     '<-' ;

RIGHTARR                                :
     '->' ;

PERIOD                                  :
     '.' ;

SEMICOLON                               :
     ';' ;

COMMA                                   :
     ',' ;

EQUALS                                  :
     '=' ;

ATSIGN                                  :
     '@' ;

LPAR                                    :
     '(' ;

RPAR                                    :
     ')' ;

LSQUARE                                 :
     '[' ;

RSQUARE                                 :
     ']' ;

LCURLY                                  :
     '{' ;

RCURLY                                  :
    '}' ;

TERM                                    :
    ~('*' | '+' | '-' | '/' | '%' | '=' | ';' | ',' | '!'
```

```
    | '|' | '<' | '>' | '{' | '}' | '[' | ']' | '(' | ')'
    | '`' | '\'' | '\"' | '\n' | '\t' | '\r' |' ')
   ~('*' | '=' | ';' | '|' | '<' | '>' | '{' | '}' |
   '[' | ']' | '(' | ')' | '`' | '\'' | '\"' | '\n' | '\t' | '\r' |' ')* ;

IDENTIFIEROFVARIABLE                        :
    '*'(ALPHANUM)+ ;

fragment ALPHANUM                           :
    '0'..'9' | 'a'..'z' | 'A'..'Z' ;
```

EK 2

```
<?xml version="1.0" encoding="utf-8"?><!DOCTYPE nta PUBLIC
'-//Uppaal Team//DTD Flat System 1.1//EN'
'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'><nta>

<declaration>
bool bieASO = false;
bool vbha2 = false;
bool gmi = false;
bool bmhNorm = false;
bool mbol = false;
bool reset = false;
bool bmha2 = false;
bool beasa2 = false;
bool vbhNorm = false;
bool beKM = false;
bool gbeAS = false;
bool gbieAS = false;
bool kguz = false;
bool gsi = false;
bool bieKM = false;
bool bieasa2 = false;
bool bieAS = false;
</declaration>
<template>
<name x="100" y="500">Beklenmedik_Mesguliyet_Hatasi_1</name>
<declaration>
</declaration>
<location id="id0" x="100" y="500">
<name x="100" y="500">D0</name>
</location>
<location id="id1" x="100" y="500">
<name x="100" y="500">D1</name>
</location>
<init ref="id0"/>
<transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500">gmi and  not gsi</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500"> not gmi and gsi</label>
</transition>
</template>
<template>
<name x="100" y="500">Beklenmedik_Mesguliyet_Hatasi_2</name>
<declaration>
```

```
</declaration>
<location id="id0" x="100" y="500">
<name x="100" y="500">BMHB</name>
</location>
<location id="id1" x="100" y="500">
<name x="100" y="500">BMH</name>
</location>
<init ref="id0"/>
<transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500"> not kguz and mbol
                                and  not gsi and gmi</label>
<label kind="assignment" x="100" y="500">bmha2:=true</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">reset</label>
<label kind="assignment" x="100" y="500">bmha2:=false</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">bmhNorm</label>
<label kind="assignment" x="100" y="500">bmha2:=false</label>
</transition>
</template>
<template>
<name x="100" y="500">Veri_Bagdasim_Hatasi</name>
<declaration>
</declaration>
<location id="id0" x="100" y="500">
<name x="100" y="500">VBHB</name>
</location>
<location id="id1" x="100" y="500">
<name x="100" y="500">VBH</name>
</location>
<init ref="id0"/>
<transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500">(gmi == gsi)</label>
<label kind="assignment" x="100" y="500">vbha2:=true</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">reset</label>
<label kind="assignment" x="100" y="500">vbha2:=false</label>
</transition>
<transition>
```

```
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">vbhNorm and (gmi != gsi)</label>
<label kind="assignment" x="100" y="500">vbha2:=false</label>
</transition>
</template>
<template>
<name x="100" y="500">Ray_Bloke</name>
<declaration>
</declaration>
<location id="id0" x="100" y="500">
<name x="100" y="500">RBB</name>
</location>
<location id="id1" x="100" y="500">
<name x="100" y="500">BE</name>
</location>
<location id="id2" x="100" y="500">
<name x="100" y="500">BIE</name>
</location>
<init ref="id0"/>
<transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500">gbeAS</label>
<label kind="assignment" x="100" y="500">beasa2:=true</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500">beKM</label>
<label kind="assignment" x="100" y="500">beasa2:=true</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">reset</label>
<label kind="assignment" x="100" y="500">beasa2:=false</label>
</transition>
<transition>
<source ref="id2"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">reset</label>
<label kind="assignment" x="100" y="500">bieasa2:=false</label>
</transition>
<transition>
<source ref="id2"/>
<target ref="id0"/>
<label kind="guard" x="100" y="500">bieASO</label>
<label kind="assignment" x="100" y="500">bieasa2:=false</label>
</transition>
<transition>
<source ref="id1"/>
```

```
<target ref="id2"/>
<label kind="guard" x="100" y="500">gbieAS</label>
<label kind="assignment" x="100" y="500">beasa2:=false</label>
<label kind="assignment" x="100" y="500">bieasa2:=true</label>
</transition>
<transition>
<source ref="id1"/>
<target ref="id2"/>
<label kind="guard" x="100" y="500">bieKM</label>
<label kind="assignment" x="100" y="500">beasa2:=false</label>
<label kind="assignment" x="100" y="500">bieasa2:=true</label>
</transition>
<transition>
<source ref="id2"/>
<target ref="id1"/>
<label kind="guard" x="100" y="500"> not gbieAS and bieAS</label>
<label kind="assignment" x="100" y="500">bieasa2:=false</label>
<label kind="assignment" x="100" y="500">beasa2:=true</label>
</transition>
</template>
<template>
<name x="100" y="500">UpdateAll</name>
<declaration>
</declaration>
<location id="id0" x="100" y="500">
<name x="100" y="500">D0</name>
</location>
<init ref="id0"/>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">bieASO:= not bieASO</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">beKM:= not beKM</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">gbeAS:= not gbeAS</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">bmhNorm:= not bmhNorm</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">gmi:= not gmi</label>
```

```
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">gbieAS:= not gbieAS</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">reset:= not reset</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">kguz:= not kguz</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">mbol:= not mbol</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">bieKM:= not bieKM</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">gsi:= not gsi</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">bieAS:= not bieAS</label>
</transition>
<transition>
<source ref="id0"/>
<target ref="id0"/>
<label kind="assignment" x="100" y="500">vbhNorm:= not vbhNorm</label>
</transition>
</template>
<system>
aUpdateAll = UpdateAll();
aBeklenmedik_Mesguliyet_Hatasi_1 = Beklenmedik_Mesguliyet_Hatasi_1();
aBeklenmedik_Mesguliyet_Hatasi_2 = Beklenmedik_Mesguliyet_Hatasi_2();
aVeri_Bagdasim_Hatasi = Veri_Bagdasim_Hatasi();
aRay_Bloke = Ray_Bloke();


        system aUpdateAll,aBeklenmedik_Mesguliyet_Hatasi_1,
                        aBeklenmedik_Mesguliyet_Hatasi_2,
```

```
                        aVeri_Bagdasim_Hatasi,aRay_Bloke;
</system>
</nta>
```

ÖZGEÇMİŞ

Ad Soyad:
Davut POLAT

Doğum Yeri ve Tarihi:
10 Ocak 1987

Adres:
Tahtakale mah. Faik Nafız Çamlıbel Cad. Olimpos Sitesi D-2-3 Blok Daire:26 Avcılar/İSTANBUL

E-Posta:
polatda@itu.edu.tr

Lisans:
2012 İstanbul Teknik Üniversitesi, Bilgisayar Mühendisliği

Lisans:
2012 İstanbul Teknik Üniversitesi, Elektronik Mühendisliği

Mesleki Deneyim ve Ödüller:
Foreks Bilgi İletişim Hizmetleri 2012-2013
Pratikkod Yazılım ve Ticaret A.Ş. 2013-2015
Foreks Bilgi İletişim Hizmetleri 2015-

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- Tolga Ovatman, Atakan Aral, Davut Polat, Ali Osman Ünver, 2014: An Overview of Model Checking Practices on Verification of PLC Software, *Journal of Software and Systems Modeling, pp. 1-24, Springer Berlin Heidelberg DOI:10.1007/s10270-014-0448-7*