SINIRSIZ PETRİ AĞLARI İÇİN TERSİNE DÖNÜŞEBİLİRLİĞİ GARANTİ EDEN SINIR VEKTÖRLERİNİN BULUNMASI

Hanife APAYDIN

Yüksek Lisans Tezi

Elektrik-Elektronik Mühendisliği

Anabilim Dalı

Aralık 2004

JÜRİ VE ENSTİTÜ ONAYI

Hanife Apaydın'ın **Sınırsız Petri Ağları için Tersine Dönüşebilirliği Garanti Eden Sınır Vektörlerinin Bulunması** başlıklı **Elektrik-Elektronik Mühendisliği** Anabilim Dalındaki Yüksek Lisans tezi 03.12.2004 tarihinde aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	İmza
Üye (Tez Danışmanı)	Yard. Doç. Dr. Aydın AYBAR	
Üye	Prof. Dr. Altuğ İFTAR	
Üye	Yard. Doç. Dr. Yusuf OYSAL	

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 12.01.2005 tarih ve 3/1... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

Prof. Dr. Altuğ İFTAR
Fen Bilimleri Enstitüsü
Müdürü

ÖZET

Yüksek Lisans Tezi

SINIRSIZ PETRİ AĞLARI İÇİN TERSİNE DÖNÜŞEBİLİRLİĞİ GARANTİ EDEN SINIR VEKTÖRLERİNİN BULUNMASI

HANİFE APAYDIN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Aydın Aybar

2004, 94 sayfa

Bu tezde sınırsız Petri ağlarında, tersine dönüşebilirlik özelliğini garanti eden bir sınır vektörünün bulunması için iki yöntem geliştirilmiştir. Yöntem 1, Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü; Yöntem 2, Petri ağının başlangıç durumuna dönen yollar (geçişler ve işaretleme vektörlerinden oluşan diziler) elde ederek, bu yollardaki işaretleme vektörlerini, dolayısıyla ağın ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü önermektedir. Bu yöntemlerden herhangi biriyle elde edilen bir sınır vektöründen yararlanarak tasarlanan bir kontrolör yardımıyla sınırsız bir Petri ağının tersine dönüşebilirliği garanti edilmektedir.

Anahtar Kelimeler: Petri ağları; Sınır vektörü; Tersine dönüşebilirlik Kapsayabilirlik ağacı; T-değişmezi.

ABSTRACT

Master of Science Thesis

DETERMINATION OF BOUND VECTORS TO GUARANTEE REVERSIBILITY FOR UNBOUNDED PETRI NETS

HANİFE APAYDIN

Anadolu University
Graduate School of Sciences
Electrical and Electronics Engineering Program

Supervisor: Assist. Prof. Aydın Aybar

2004, 94 pages

In this thesis, two methods yielding bound vectors which guarantee reversibility of an unbounded Petri net are developed. Method 1 proposes a bound vector that covers all of the marking vectors in a reversible subset of the reachability set of the Petri net. Method 2 finds loops including initial marking of the Petri net and proposes a bound vector that covers all of the vectors in these loops, consequently it covers all of the marking vectors in a reversible subset of the reachability set of the Petri Net. Reversibility of an unbounded Petri net is guaranteed by a controller designed by using a bound vector which is determined by one of these methods.

Keywords: Petri nets; Bound vector; Reversibility; T-invariant; Incidence matrix.

TEŞEKKÜR

Çalışmalarım boyunca beni destekleyen, bilgi ve tecrübelerini benimle paylaşan sayın hocam Yard. Doç. Dr. Aydın Aybar'a, ayrıca tez savunmama katılarak destek olan, yardım ve önerilerini esirgemeyen hocalarım sayın Prof.Dr. Altuğ İftar'a ve sayın Yard. Doç. Dr. Yusuf Oysal'a katkılarından dolayı teşekkürlerimi sunarım.

Değerli aileme ve eşim Erkan Özkan'a sonsuz sabır ve desteklerinden dolayı, arkadaşlarım Ümmühan Başaran ve Nuray At'a her türlü yardımlarından dolayı teşekkür ederim.

Hanife Apaydın

Aralık 2004

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

SİMGELER DİZİNİ

D : Doğal sayılar kümesi

D^+ : Sayma sayılar kümesi

$|A|$: A kümesinin eleman sayısı

B^T : B matrisinin transpozu

m_0 : Başlangıç işaretleme vektörü

$\tilde{P}$: Petri ağında sınırsız yerler kümesi

R' : Başlangıç işaretleme vektörüne yalnızca bir geçiş ateşlemesiyle ulaşabilen işaretleme vektörleri kümesi

w : Kapsayabilirlik ağacında sınırsız sayıda işaretleme vektörü oluşumunu engellemek için bazı yerlerin belirti sayısının ifadesinde kullanılan sembol

φ_1 : G ile gösterilen bir Petri ağının kapsayabilirlik ağacında bulunan ve w bulundurmayan işaretleme vektörleri kümesi

φ_2 : G ile gösterilen bir Petri ağının kapsayabilirlik ağacında bulunan ve w bulunduran işaretleme vektörleri kümesi

φ : G ile gösterilen bir Petri ağının kapsayabilirlik ağacında bulunan tüm işaretleme vektörlerinin oluşturduğu küme

$\mathcal{O}$: sıfır vektörü

$E(G, M)$: G ile gösterilen bir Petri ağında, M işaretleme vektöründen ateş lenebilen geçişlerin oluşturduğu küme

$R(G, M)$: G ile gösterilen bir Petri ağında, M işaretleme vektöründen sonra ulaşılabilecek işaretleme vektörlerinin oluşturduğu küme

1. GİRİŞ

Teknoloji ilerledikçe sanayide kullanılan sistemler büyümüş, gelişmiş ve karmaşıklaşmıştır. Dolayısıyla, sistemlerin en uygun biçimde modellenmesi ve kontrolü gün geçtikçe önemini arttırmaktadır.

Günümüzde büyük ölçekli endüstriyel üretim süreçlerinin çeşitli amaçlar doğrultusunda izlenmesi ve denetimine ilişkin tasarımlar ve gerçeklemelerde kesikli olay sistemler kullanılmaya başlanmıştır [1]. Sözkonusu sistemler olay etkileşimli (event-driven) olarak da adlandırılırlar. Birçok sistemi (otomatik üretim sistemleri, haberleşme ağları, ofis bilgi sistemleri vb.) kapsamına alan kesikli olay sistemlerindeki olayların meydana gelişi, diğer olay ya da olaylara bağlı olduğundan modellemede çok fazla değişken kullanmak gerekmektedir. Bundan dolayı, kesikli olay sistemlerin modellenmesinde kullanılmak üzere Markov zincirleri, minimum-maksimum cebir modelleri, Petri ağları gibi çeşitli modelleme yöntemleri geliştirilmiştir [2].

Petri ağı ile eş zamanlı olmayan paralel olaylar arasındaki ilişki ve olayların birbirine olan etkileşimleri ortaya konularak, karmaşık sistemlerin analizine olanak sağlanmıştır. Petri ağları en çok, bazı olayların birbirinden bağımsız olarak meydana gelebildiği sistemlerin (haberleşme protokolleri, endüstriyel kontrol sistemleri vb.) modellenmesinde kullanılmaktadır [3].

Petri ağı ilk kez Carl Adam Petri tarafından ortaya konmuştur [1]. O zamandan beri eş zamanlı olmayan birbirinden bağımsız (concurrent) sistemlerin analizi ve modellenmesi için kullanılan ağ teorileri üzerindeki çalışmalar devam etmektedir. Petri ağı modellemesinin zamanlanmış Petri ağı, renklendirilmiş Petri ağı gibi birçok çeşidi geliştirilmiştir [4]. Bu çalışmada temel Petri ağı modellemesi ele alınmıştır.

Petri ağı ile modellenmiş bir sistemin birçok özelliğinin (tersine dönüşebilirlik, sınırlılık vb.) analizi için kullanılan kapsayabilirlik (coverability) ağacı ve ulaşılabilirlik (reachability) ağacı gibi araçlar ve bu araçlar arasındaki ilişki ilk kez Karp ve Miller [5] tarafından ortaya konmuştur. Petri ağının

bağlantı yapısına bağlı özelliklerinin analizi için kullanılan çakışım matrisi (incidence matrix) ile T değişmezleri (T invariants) ve P değişmezleri (P invariants) Sifakis [6] tarafından sunulmuştur. [7] ve [8]'de yapılan çalışmalarda da bu değişmezlerin bulunmasını sağlayan algoritmalar geliştirilmiştir.

Petri ağlarının özelliklerinin analizi için, Ye ve ark. [9] tarafından yapılan çalışmada, ulaşılabilirlik ağacı ile Petri ağının korunumluluk, sınırlılık, güvenlilik analizleri yapılmıştır.

Aybar, İftar ve Apaydın [10] tarafından, sınırsız Petri ağları için sınırlılığı, tersine dönüşebilirliği ve canlılığı garanti eden merkezi ve dışmerkezli kontrol yaklaşımları sunulmuştur. İlgili çalışmada, sınırlılık özelliği için uygun bir sınır vektörü kullanılmıştır.

Bir Petri ağında m_0'dan ulaşılabilen tüm durumlar yeniden m_0'a ulaşabiliyorlar ise Petri ağı tersine dönüşebilirdir. Eğer Petri ağında m_0'dan ulaşılabilen m_0'dan farklı en az bir durum, yeniden m_0'a ulaşabiliyorsa Petri ağı kısmi tersine dönüşebilirdir. Bir Petri ağı kısmi tersine dönüşebilir ise, bu ağın ulaşılabilirlik kümesinin kısmi tersine dönüşebilir bir alt kümesi vardır ve bu küme ağın ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesini içerir. [11]'de yapılan çalışmada geliştirilen algoritmalar ile daha önceden belirlenmiş bir sınır vektörü kullanılarak, sınırsız bir Petri ağının ulaşılabilirlik kümesinin sınırlı sayıda elemandan oluşan bir alt kümesi elde edilmiş ve bu kümenin kısmi tersine dönüşebilir olması durumunda, tersine dönüşebilir kısmının bulunması sağlanmıştır.

Bu tezde amaç; sınırsız, temel bir Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesini oluşturmak için bir sınır vektörü bulmaktır. Bu amaç doğrultusunda iki yöntem geliştirilmiştir. Yöntem 1 ile, Petri ağının kısmi tersine dönüşebilirlik analizi yapılmakta, ağ kısmi tersine dönüşebilir ise ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü önerilmektedir. Yöntem 2 ile de Petri ağının belirli T-değişmezleri kullanılarak başlangıç durumuna dönen yollar (geçişler ve işaretleme vektörlerinden oluşan bir dizi) bulunmakta, bu yollar-

daki işaretleme vektörlerini ve dolayısıyla ağın ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü önerilmektedir. Bu yöntemlerden herhangi biriyle elde edilen bir sınır vektöründen yararlanarak tasarlanan bir kontrolör yardımıyla, Petri ağının tersine dönüşebilirliği garanti edilmektedir. Ayrıca bu tezde, Petri ağının kapsayabilirlik ağacı, T değişmezi ve geliştirilen iki yöntem için Matlab programları oluşturulmuştur.

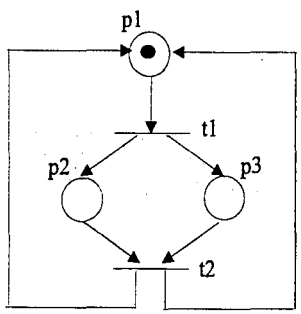
Bu çalışmada, Petri ağı modeli, bu modelin matematiksel ifadesi ve özellikleri Bölüm 2'de, Petri ağı için analiz yöntemleri Bölüm 3'de verilecektir. Bölüm 4'de, geliştirilen iki yöntem ve oluşturulan algoritmalar anlatılacaktır. Geliştirilen yöntemler için oluşturulan Matlab programlarının farklı Petri ağları üzerinde uygulanması ile elde edilen sonuçlar Bölüm 5'de verilecektir.

2. PETRİ AĞLARI

Bu bölümde, [1, 4, 12]'deki çalışmalar temel alınarak Petri ağının tanımı, çalışması ve özellikleri verilecektir.

Bir Petri ağı, daire şeklindeki yerler, çubuk şeklindeki geçişler ve bunlar arasındaki bağlantıyı sağlayan yönlendirilmiş oklardan oluşur. Yer, bir işlemi veya bir kaynağın durumunu; geçiş ise, bir olayın ya da bir işlemin, başlangıcını ve/veya bitişini göstermektedir[1].

Örneğin, Şekil 2.1'deki Petri ağı üç yerden (p_1, p_2, p_3) ve iki geçişten (t_1, t_2) oluşmaktadır. p_1, t_1 geçişinin girdisi, p_2 ve p_3, t_1 geçişinin çıkışı; benzer şekilde t_1, p_2 yerinin girdisi, t_2 aynı yerin çıkışı olarak verilmiştir.



Şekil 2.1: Bir Petri ağı

Bir Petri ağının yapısında yerler ve geçişlere ek olarak bulunduğu yerlerde işlemin yapılıyor olduğunu ifade eden " . " şeklinde gösterilen belirtiler vardır. Belirtinin olup olmamasına veya sayısına göre ilgili yerlerin gösterdikleri kaynak hakkında bilgi sahibi olunur. Burada kaynak, sistemde gerçekleştirilmek istenen olaylar için gerekli olan koşulları, çalışan personeli, kullanılan makineleri vb. ifade etmektedir. Örneğin, bir yer kaynağı ifade ediyorsa, bu yerin içinde belirti olması kaynağın erişilebilir olduğunu, belirti sayısı da erişilebilir kaynak sayısını gösterir. Belirtiler mantıksal durumu ifade

[1]Bu tezde, aynı yer ile aynı geçiş arasında aynı yönlü en fazla bir ok olacağı kabullenmesi yapılmıştır.

etmek için de kullanılır. Bir yerde belirti bulunması şartın sağlandığı bulunmaması ise şartın sağlanmadığı anlamına da gelir.

Bir Petri ağı $\mathbf{G}(P, T, N, O, m_0)$ şeklinde gösterilir. Burada T, geçişlerin kümesini; P, yerlerin kümesini göstermektedir ($P \cap T = \emptyset$, $P \cup T \neq \emptyset$). N: $P \times T \rightarrow \{0, 1\} =: \delta$ girdi matrisidir. Bu matris yerlerden geçişlere doğru olan bağlantıların gösterildiği matristir ve elemanları, bir yerden bir geçişe bir ok ile bağlantı yapılmışsa 1, yapılmamışsa 0 alınarak oluşturulur. O: $P \times T \rightarrow \delta$ çıktı matrisidir. Bu matris geçişlerden yerlere doğru olan bağlantıların gösterildiği matristir ve elemanları bir geçişten bir yere bir ok ile bağlantı yapılmışsa 1, yapılmamışsa 0 alınarak oluşturulur. m_0: $P \rightarrow D$ başlangıç işaretleme vektörüdür. Başlangıçta, belirtilerin Petri ağının yerlerindeki dağılımını gösterir. Ayrıca, Petri ağının ifadesinde N ve O girdi matrislerinin yerine $A := O - N$ şeklinde tanımlanan çakışım matrisi (incidence matrix) de kullanılabilir [12].

Örneğin, Şekil 2.1'deki Petri ağı için, P=$\{p_1, p_2, p_3\}$, T=$\{t_1, t_2\}$, m_0=$[1\ 1\ 0]^T$ ve

$$N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \qquad O = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \qquad A = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}$$

şeklindedir.

Bir Petri ağının çalışması geçişlerin ateşlenmesiyle belirtilerin yer veya yerlerden başka yer veya yerlere taşınması şeklinde olur. Bir Petri ağının yapısında bulunan bir okun bağlantılı olduğu geçişin ateşlenmesi ile taşıyabileceği belirti sayısına bu okun ağırlığı denir. Yapısındaki tüm okların ağırlığı 1 olan Petri ağı temel (ordinary) Petri ağı olarak isimlendirilir [12].

M bir işaretleme vektörünü göstermek üzere, bu işaretleme vektöründen sonra bir $t \in T$ geçişinin ateşlenebilirlik şartı :

$$M(p) \geq N(p, t), \forall p \in P \tag{2.1}$$

şeklindedir. Burada, $M(p)$, M işaretleme vektörünün p yerindeki belirti sayısını göstermektedir.

Bir M işaretleme vektöründen bir t geçişinin ateşlenmesiyle bir M' işaretleme vektörü oluşuyorsa bu matematiksel olarak,

$$M' = M + O_t - N_t,\ t \in T \tag{2.2}$$

şeklinde ifade edilir. Burada, N_t, N matrisinin t geçişine karşılık gelen sütunu O_t, O matrisinin t geçişine karşılık gelen sütunudur.

Petri ağında peşpeşe ateşlenen geçişlerin oluşturduğu diziye ateşleme dizisi veya geçiş dizisi denir. Bir g ateşleme dizisi $g := t_{i_1},\ t_{i_2}.....t_{i_k}: t_{i_1}.t_{i_2}...$ $...t_{i_k} \in T$ şeklinde gösterilir. Bu g ateşleme dizisindeki bütün geçişlerin sırasıyla (t_{i_1} geçişinden başlayarak en son t_{i_k} geçişinin) ateşlenmesiyle herhangi bir M işaretleme vektöründen bir M' işaretleme vektörü elde edilebiliyor ise bu M'=$\rho(M, g)$ ile ifade edilir. Burada kullanılan kısmi fonsiyon, $\rho(M, g)$, iletim fonksiyonu olarak adlandırılır.

Ulaşılabilirlik kümesi, $R(G, m_0)$, G ile gösterilen bir Petri ağında, m_0 işaretleme vektöründen ulaşılan tüm işaretleme vektörlerinin oluşturduğu kümedir. Bu tezin tamamında temel ve $R(G, m_0)$ kümesinin eleman sayısının en az iki olması şartını sağlayan ($|R(G, m_0)| \geq 2$) Petri ağları ele alınmıştır.

2.1 Petri Ağlarının Özellikleri

Bu bölümde Petri ağlarının bazı özelliklerinin tanımları [12, 13]'den yararlanılarak verilecektir.

Tanım 1.1 [Ulaşılabilirlik (Reachability)]: Eğer ve yalnızca eğer $\rho(m_0, g) = M_r \in R(G, m_0)$ olacak şekilde bir g ateşleme dizisi bulunabiliyorsa M_r işaretleme vektörü m_0'dan ulaşılabilirdir.

Tanım 1.2 [Sınırlılık (Boundedness)]: Eğer ve yalnızca eğer $\forall M \in R(G, m_0)$. $M(p) \leq \hat{k}$ olacak şekilde $\hat{k} \in D$ bulunabiliyorsa p yeri $\hat{k}$ ile sınırlıdır. Eğer ve

yalnızca eğer $M(p) \leq k(p)$, $(k = [\hat{k}_1\ \hat{k}_2....\hat{k}_{|P|}]^T)$, $\forall p \in P$, $\forall M \in R(G, m_0)$ ise bu ağ k-sınırlıdır.

Tanım 1.3 [Canlılık (Liveness)]: Eğer ve yalnızca eğer $\forall M \in R(G, m_0)$ işaretlemesinden sonra t $(\forall t \in T)$ geçişine izin verecek şekilde bir g ateşleme dizisi varsa Petri ağı canlıdır.

Tanım 1.4 [Tersine Dönüşebilirlik (Reversibility)]: Eğer ve yalnızca eğer $\forall M \in R(G, m_0)$, $m_0 \in R(G, M)$ şartı sağlanıyorsa bu Petri ağı tersine dönüşebilirdir.

Tanım 1.5 [Kısmi Tersine Dönüşebilirlik (Partial Reversibility)]: Eğer ve yalnızca eğer en az bir $M \in R(G, m_0)$ ve $m_0 \neq M$ için $m_0 \in R(G, M)$ şartı sağlanıyorsa bu Petri ağı kısmi tersine dönüşebilirdir.

Tanım 1.6 [Kapsayabilirlik (Coverability)]: Eğer ve yalnızca eğer M, $\tilde{M} \in R(G, m_0)$ olmak üzere $\tilde{M}(p) \geq M(p)$, $\forall p \in P$ ise $\tilde{M}$ işaretleme vektörü M işaretleme vektörünü kapsar.

Tanım 1.7 [Baskınlık (Dominance)]: Eğer ve yalnızca eğer M, $\tilde{M} \in R(G, m_0)$ olmak üzere, $\tilde{M}(p) \geq M(p)$, $\forall p \in P$, $\tilde{M} \neq M$ ise $\tilde{M}$ işaretleme vektörü M işaretleme vektörüne baskındır ve bu $\tilde{M} >_d M$ ile gösterilir.

Tanım 1.8 [Çıkmaz işaretleme vektörü (Deadlock marking vector)]: Eğer ve yalnızca eğer, bir Petri ağında herhangi bir $M \in R(G, m_0)$ işaretlemesinden sonra hiçbir $t \in T$ geçişi ateşlenemiyorsa, M çıkmaz işaretleme vektörü olarak adlandırılır. Bu durumda Petri ağında sistem çıkmazı (deadlock) meydana gelmiş olur.

Tanım 1.9 [Cansız geçiş (Dead transition)]: Petri ağının yapısında görülen, ancak ağın çalışması esnasında hiçbir şekilde ateşlenmeyen geçişe cansız geçiş denir.

Yukarıda tanımları verilen özellikler, takip eden bölümlerde kullanılacaktır.

3. PETRİ AĞLARI İÇİN ANALİZ YÖNTEMLERİ

Petri ağı ile modellenmiş bir sistemin Bölüm 2'de verilen özelliklerinin analiz edilmesi için birçok yöntem geliştirilmiştir [4]. Bu bölümde kapsayabilirlik ağacı yöntemi ve ağ yapısındaki geçiş ve yerlerin birbirine bağlantısına göre oluşturulan çakışım matrisi yöntemi üzerinde durulacaktır.

3.1 Kapsayabilirlik Ağacı

Kapsayabilirlik ağacı, Petri ağının ulaşılabilirlik kümesinin grafiksel olarak gösterimidir. İlk olarak başlangıç işaretleme vektörü ağacın en üstüne yerleştirilir, daha sonra bundan ulaşılabilecek tüm işaretleme vektörleri, ateşlenebilecek tüm geçişlerin ateşlenmesiyle elde edilerek sırayla bu yapıya eklenir.

Kapsayabilirlik ağacının oluşturulması için [12]'de sunulan algoritma aşağıda verilmiştir. Bu algoritmada *çıkmaz*, çıkmaz işaretleme vektörlerine, *eski* daha önce elde edilmiş işaretleme vektörlerine verilen etikettir.

$[\varphi]$=**Algoritma-1**$[G]$

1. m_0, ağacın en üst noktasına yerleştirilir.
2. Ağaç yapısındaki *eski* veya *çıkmaz* etiketli olmayan işaretleme vektörleri sırayla seçilerek herbiri için şu işlemler yapılır: (seçilen işaretleme vektörü M olarak gösterilsin)
   - 2.1. Eğer M işaretleme vektöründen hiçbir geçiş ateşlenemiyorsa *çıkmaz* olarak etiketlenir, 2. adıma dönülür.
   - 2.2. Eğer M, ağaç yapısındaki herhangi bir işaretleme vektörüne eşitse *eski* olarak etiketlenir, 2. adıma dönülür.
   - 2.3. Diğer durumlarda M işaretleme vektöründen ateşlenebilen tüm geçişler sırayla seçilip, herbiri için aşağıdaki işlemler yapıldıktan sonra

2. adıma dönülür.

a-) Geçiş ateşlenerek yeni bir işaretleme vektörü elde edilir (bu işaretleme vektörü M' olsun).

b-) m_0'dan M' işaretleme vektörüne kadar olan yolda (geçişler ve yerlerden oluşan bir dizi) $M' >_d M''$. ifadesini sağlayacak bir M'' işaretleme vektörü varsa $M'(p) > M''(p)$ olan her yerde $M'(p)$, w olarak değiştirilir. M' ateşlenen geçişle birlikte ağaca yerleştirilir.

3. Kapsayabilirlik ağacında bulunan tüm işaretleme vektörleri ile φ. kapsayabilirlik ağacında bulunan ve hiçbir yerinde w bulundurmayan işaretleme vektörleri ile φ_1, kapsayabilirlik ağacında bulunan ve herhangi bir yerinde w bulunduran işaretleme vektörleri ile φ_2 kümeleri oluşturulur ($\varphi_2 \cup \varphi_1 = \varphi$, $\varphi_2 \cap \varphi_1 = \emptyset$).

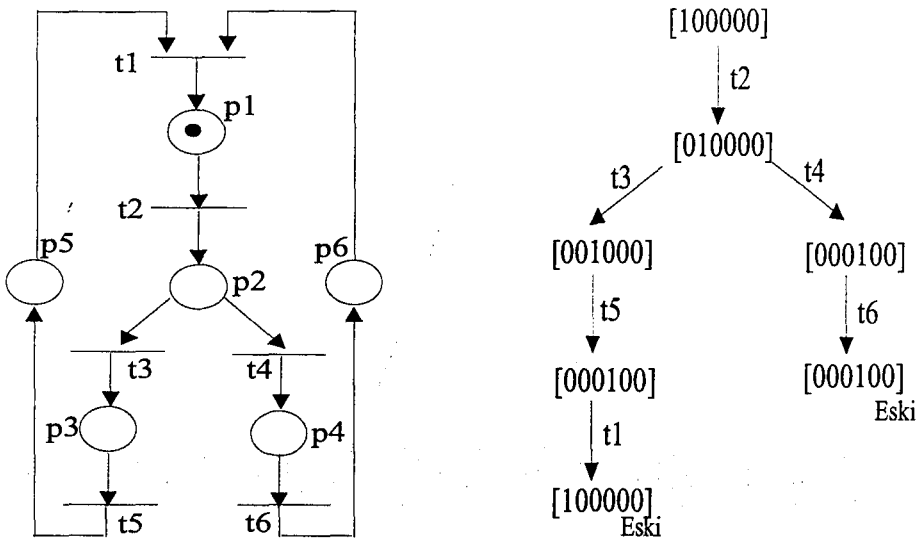
Kapsayabilirlik ağacı oluşturulurken, sınırsız sayıda işaretleme vektörünün meydana gelmesini engellemek için sınırsız yerlerdeki belirti sayısının ifadesinde w sembolü kullanılmaktadır.

Şekil 3.1'de sınırlı bir Petri ağı ve bu ağ için yukarıdaki yöntem kullanılarak oluşturulan kapsayabilirlik ağacı, Şekil 3.2'de ise sınırsız bir Petri ağı ve bu ağ için yine yukarıdaki yöntem kullanılarak oluşturulan kapsayabilirlik ağacı görülmektedir.

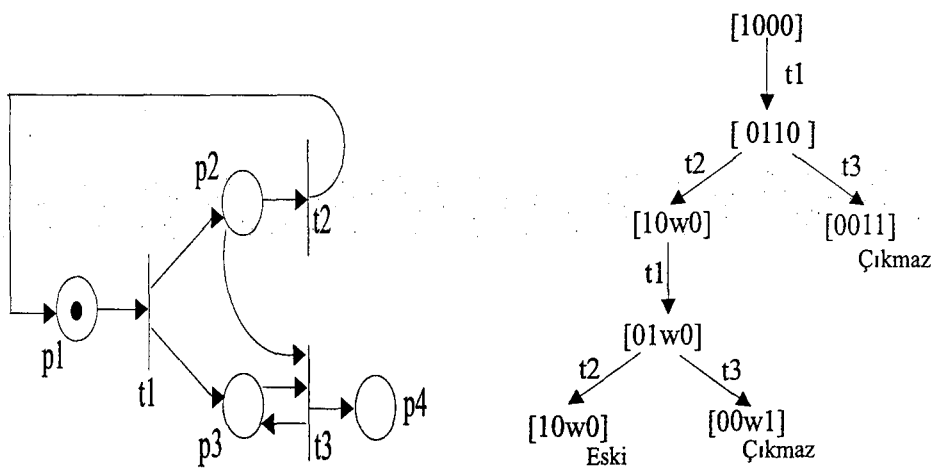
Yapısında, w içeren işaretleme vektörleri bulunan bir kapsayabilirlik ağacına sahip bir Petri ağının ulaşılabilirlik kümesinin tamamını, kapsayabilirlik ağacında görmek mümkün değildir. Ancak, hiçbir yerinde w bulundurmayan işaretleme vektörlerinden oluşan bir kapsayabilirlik ağacı, ait olduğu Petri ağının ulaşılabilirlik kümesini ifade eder. Bu durumda Şekil 3.1'deki Petri ağı için $\varphi = \varphi_1 = R(G, m_0)$ ={$[1\ 0\ 0\ 0\ 0\ 0]^T$, $[0\ 1\ 0\ 0\ 0\ 0]^T$, $[0\ 0\ 1\ 0\ 0\ 0]^T$, $[0\ 0\ 0\ 1\ 0\ 0]^T$, $[0\ 0\ 0\ 0\ 1\ 0]^T$, $[0\ 0\ 0\ 0\ 0\ 1]^T$}, $\varphi_2 = \emptyset$ şeklindedir.

Şekil 3.2.a'daki Petri ağı için oluşturulan kapsayabilirlik ağacından faydalanılarak (bkz. Şekil 3.2.b), bu Petri ağı için φ={$[1\ 0\ 0\ 0]^T$, $[0\ 1\ 1\ 0]^T$,

$[1\ 0\ w\ 0]^T$, $[0\ 0\ 1\ 1]^T$, $[0\ 1\ w\ 0]^T$, $[1\ 0\ w\ 0]^T$, $[0\ 0\ w\ 1]^T\}$. $\varphi_1 = \{[1\ 0\ 0\ 0]^T$, $[0\ 1\ 1\ 0]^T$, $[0\ 0\ 1\ 1]^T\}$, φ_2=$\{[1\ 0\ w\ 0]^T$, $[0\ 1\ w\ 0]^T$, $[1\ 0\ w\ 0]^T$, $[0\ 0\ w\ 1]^T\}$ şeklinde belirlenir. $[0\ 0\ 1\ 1]^T$ ve $[0\ 0\ w\ 1]^T$ işaretleme vektörlerinden hiçbir geçiş ateşlenemez. Dolayısıyla bunlar bu ağdaki çıkmaz işaretleme vektörleridir. Bu ağ için. $T = \{t_1, t_2, t_3\}$ (ağın modellemesinde görülen geçişler) olup kapsayabilirlik ağacı oluşurken hepsi ateşlenmektedir. Petri ağında cansız geçiş yoktur.



Şekil 3.1: (a) Petri ağı-1 ve (b) Petri ağı-1 için kapsayabilirlik ağacı



Şekil 3.2: (a) Petri ağı-2 ve (b) Petri ağı-2 için kapsayabilirlik ağacı

Kapsayabilirlik ağacını meydana getirmek için verilen yöntem kullanılarak MATLAB'da oluşturulan Cover.m adlı program Ek-A'da bulunmaktadır.

Cover.m programının Şekil 3.2.a'daki Petri ağı için oluşan çıktı dosyasında (bkz. Şekil 3.3) Bmat, Resmat, Txmat ve Oldmat başlıkları altındaki bilgiler kapsayabilirlik ağacının yapısını ifade eder. Bmat başlığının altındaki ilk satırda m_0 bulunur. m_0'dan Txmat başlığı altındaki ilk geçiş ateşlenerek Resmat'ın altındaki ilk satırda bulunan işaretleme vektörü oluşur. Bu vektör daha önce oluştuysa, kapsayabilirlik ağacındaki *eski* etiketli bir işaretleme vektörünü ifade eder ve Oldmat'ın altındaki ilk satırda 1, oluşmadıysa Oldmat'ın altındaki ilk satırda 0 bulunur. m_0'dan başka bir geçiş daha ateşlenebiliyorsa; m_0, Bmat'ın ikinci satırına da yerleştirilir. Resmat, Txmat ve Oldmat'ın ikinci satırları da birinci satırda olduğu gibi oluşturulur. m_0'dan ateşlenebilen tüm geçişler için aynı işlemler yapılır. Daha sonra yeni oluşan ve kapsayabilirlik ağacındaki *eski* etiketli bir işaretleme vektörünü ifade etmeyen işaretleme vektörleri sırasıyla Bmat'a yerleştirilir ve m_0 için yapılan işlemler bu vektörler için de yapılır. Kendisinden ateşleme yapılabilecek bir işaretleme vektörü (kapsayabilirlik ağacındaki *eski* veya *çıkmaz* etiketli bir işaretleme vektörünü ifade etmeyen bir işaretleme vektörü) kalmayana kadar bu işlemlere devam edilir.

Örneğin, Şekil 3.2'deki Petri ağının kapsayabilirlik ağacı için, Şekil 3.3'de $[1\ 0\ 0\ 0]^T$'dan t_1 ateşlenerek $[0\ 1\ 1\ 0]^T$ meydana gelmekte, bu işaretleme vektörü daha önce oluşmadığı için Oldmat'ın bu satırında 0 bulunmaktadır. $[1\ 0\ 0\ 0]^T$'dan başka geçiş ateşlenememektedir. $[0\ 1\ 1\ 0]^T$ daha önce oluşmadığından, Bmat'a yerleştirilmekte, bu vektörden ikinci satırda t_2 nin ateşlenmesiyle $[1\ 0\ w\ 0]^T$, üçüncü satırda, t_3 ün ateşlenmesiyle $[0\ 0\ 1\ 1]^T$ oluşmaktadır. $[1\ 0\ w\ 0]^T$ Bmat'a yerleştirilerek aynı işlemler bu vektör için de tekrarlanmaktadır. $[0\ 0\ 1\ 1]^T$ çıkmaz işaretleme vektörü olduğundan Bmat'da bulunmayacaktır. Bu işlemler kendisinden ateşleme yapılabilecek bir işaretleme vektörü kalmayana kadar devam etmekte ve kapsayabilirlik ağacı

inşaası tamamlanmaktadır.

Bmat	Resmat	Ixmat	Oldmat
1000	0110	1	0
0110	10w0	2	0
0110	0011	3	0
10w0	01w0	1	0
01w0	10w0	2	1
01w0	00w1	3	0

ÇIKMAZ VEKTÖR
0011
00w1

CANSIZ GEÇIS
YOK

Şekil 3.3: Şekil 3.2'deki Petri ağı için Cover.m programının çıktı dosyası

3.2 Kapsayabilirlik Ağacı ile Petri Ağı Analizi

G ile ifade edilen bir Petri ağının kapsayabilirlik ağacı kullanılarak, bu ağın birçok özelliği hakkında bilgi edinilmektedir (bkz. [12, 13, 4]). Bunlardan bazıları bu bölümde özetlenecektir.

Eğer ve yalnızca eğer bir Petri ağının kapsayabilirlik ağacındaki hiçbir işaretleme vektöründe w görünmüyorsa bu Petri ağı sınırlıdır. Bu durumda $\varphi = R(G, m_0)$ şeklinde elde edilmektedir. Kapsayabilirlik ağacındaki herhangi bir işaretleme vektöründe w bulunması bu ağın sınırsız olduğunu, ve bu sembolün bulunduğu yerin de sınırsız bir yer olduğunu gösterir. Kapsayabilirlik ağacındaki işaretleme vektörlerinin hiçbirinde w içermeyen yerler, sınırlı yerlerdir.

Eğer bir Petri ağının kapsayabilirlik ağacında bulunan herbir işaretleme vektöründen m_0'a dönen bir yol (geçişlerden ve işaretleme vektörlerinden oluşan bir dizi) bulunabiliyorsa bu ağ tersine dönüşebilirdir. Eğer bir Petri ağının kapsayabilirlik ağacında, m_0'dan farklı en az bir işaretleme vektöründen

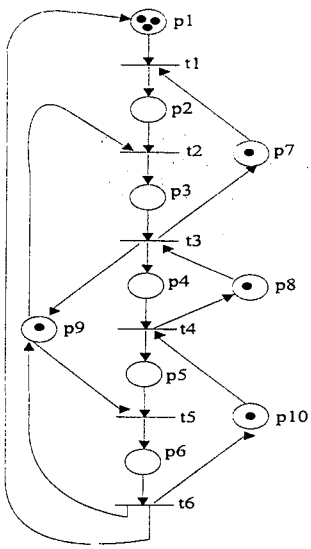
m_0'a dönen bir yol bulunabiliyorsa bu Petri ağı kısmi tersine dönüşebilirdir ve ağın ulaşılabilirlik kümesinin tersine dönüşebilir özellikte bir alt kümesi vardır. Sınırsız bir ağ için tersine dönüşebilirlik özelliğini, sınırsız yerlerde w gösterimi nedeniyle, kapsayabilirlik ağacından takip etmek mümkün değildir. Sınırsız bir Petri ağının ilgili özelliğinin analizi için kapsayabilirlik ağacı kullanılarak oluşturulan yöntemden Bölüm 4'de bahsedilecektir.

Sınırsız Petri ağlarının kapsayabilirlik ağacında w sembolü bulunduğundan ve bu sembol özellikle yerlerde bulunan belirti sayısı ile ilgili bilgi kaybına sebep olduğundan, buna ek olarak ulaşılabilirlik kümesi sonsuz sayıda işaretleme vektörü içerdiğinden, canlılık ve ulaşılabilirlik özellikleri kapsayabilirlik ağacı ile kolayca analiz edilemez. Buna rağmen şu yaklaşımlar yapılmaktadır:

- Kapsayabilirlik ağacında *çıkmaz* olarak isimlendirilmiş en az bir işaretleme vektörü bulunan ağ canlı değildir [14].
- Kapsayabilirlik ağacının herhangi bir işaretleme vektörü tarafından kapsanmayan bir işaretleme vektörü (bkz. Tanım 1.6) m_0' dan ulaşılabilir değildir [13].

Örnek 3.1

Şekil 3.4'deki Petri ağı için oluşturulan kapsayabilirlik ağacı, Şekil 3.5'deki gibidir. Bu kapsayabilirlik ağacında hiçbir işaretleme vektöründe w bulunmadığı için bu Petri ağı sınırlıdır. Kapsayabilirlik ağacındaki işaretleme vektörlerinde, yerlerde bulunan en büyük belirti sayısından yararlanılarak bu Petri ağı için sınır vektörü $[3\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^T$ şeklinde belirlenir. Çıkmaz işaretleme vektörünün ($[0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0]^T$) varlığından dolayı, bu Petri ağı tersine dönüşebilir veya canlı değildir.



Şekil 3.4: Örnek 3.1 için bir Petri ağı

Bmat	Resmat	Txmat	Oldmat
3000001111	2100000111	1	0
2100000111	2010000101	2	0
2010000101	2001001011	3	0
2001001011	1101000011	1	0
2001001011	2000101110	4	0
1101000011	1011000001	2	0
1101000011	1100100110	4	0
2000101110	1100100110	1	1
2000101110	2000011100	5	0
1011000001	1010100100	4	0
1100100110	1010100100	2	1
1100100110	1100010100	5	0
2000011100	1100010100	1	1
2000011100	3000001111	6	1
1010100100	1001101010	3	0
1100010100	2100000111	6	1
1001101010	0101100010	1	0
1001101010	1001011000	5	0
0101100010	0011100000	2	0
0101100010	0101010000	5	0
1001011000	0101010000	1	1
1001011000	2001001011	6	1
0101010000	1101000011	6	1

ÇIKMAZ VEKTÖR
0011100000

CANSIZ GEÇIS
YOK

Şekil 3.5: Örnek 3.1 için Cover.m programının çıktı dosyası

3.3 Çakışım Matrisi

Bu bölümde Petri ağı analizinde kullanılan, doğrusal matris cebirine dayalı çakışım matrisi yöntemi üzerinde durulacaktır. Bu analizin kapsayabilirlik ağacı yöntemine göre en büyük avantajı ağın analizi için basit doğrusal cebir denklemlerinin kullanılmasının yeterli olabilmesidir [12].

Bir M işaretleme vektöründen bir g geçiş dizisinin ateşlenmesiyle bir M' işaretleme vektörünün elde edilmesi,

$$M' = M + AU \tag{3.1}$$

ile ifade edilir. Burada $U := T \rightarrow D$, g ateşleme dizisini meydana getiren geçişlerin bu dizi içinde kaç kez kullanıldığını gösteren vektördür [1] ve g geçiş dizisinin ateşleme sayısı vektörü olarak isimlendirilir. Örneğin, Şekil 3.2'deki Petri ağı için ($T = \{t_1, t_2, t_3\}$) $g = t_1 t_2$ şeklinde tanımlanan bir geçiş dizisinin ateşleme sayısı vektörü, $U = [1\ 1\ 0]^T$ olarak belirlenir.

3.3.1 T Değişmezi

T değişmezi, çakışım matrisi kullanılarak elde edilen, Petri ağının m_0'a bağlı olmayan, sadece bağlantı yapısıyla ilgili özelliklerinin (yapısal özellikler, [13]) analizinde kullanılan bir vektördür.

Tanım 3.1: W negatif elemanı bulunmayan $|T| \times 1$ boyutunda, $W(i) \in D$, $\forall i \in \{1, 2, ...|T|\}$ olan ve

$$AW = \mathcal{O} \tag{3.2}$$

şartlarını sağlayan bir vektörse, bu vektör Petri ağının T-Değişmezidir [13].

Eşitlik 3.1'de ateşleme sayısı vektörü yerine W koyulduğunda,

$$M' = M + AW \tag{3.3}$$

eşitliğin sağındaki ikinci eleman $\mathcal{O}$ olacağından $M' = M$ elde edilecektir. Bu durumda, $W \neq \mathcal{O}$ şeklindeki bir T-Değişmezine karşılık gelen geçiş dizileri ateşlenebilmeleri durumunda, sistemi ateşlemenin başladığı duruma geri döndürecektir.

Örneğin, T-değişmezi $[0\ 1\ 1\ 1]^T$ ise bu değişmeze karşılık gelen g geçiş dizileri t_2, t_3 ve t_4 geçişlerini birer kez kullanan dizilerdir ($t_2t_3t_4$, $t_3t_2t_4$, $t_4t_2t_3$... vb.). Bu dizilerden herhangi biri, örneğin $g = t_4t_3t_2$, bir M işaretleme vektöründen ateşlenebiliyorsa, $\rho(M, g) = M$ şeklinde olacaktır.

Eğer rank(A)=$|T|$ ise, bu Petri ağının $W = \mathcal{O}$'dan farklı bir T-değişmezi yoktur (rank(A), A matrisinin rankını göstermektedir) ([12]).

3.3.2 T-Değişmezinin Bulunması

$AW = \mathcal{O}$ denkleminin bilinen doğrusal matris cebiri metodlarıyla çözümü oldukça kolay bulunabilmektedir [15]. Ancak T-değişmezi bulunurken aranan şartlar ($W(i) \in D$, $\forall i \in \{1, 2, ...|T|\}$) bu çözümü karmaşık bir hale sokmaktadır. [12]'de bir Petri ağının T-değişmezlerinin bulunması için önerilen Mark&Silva Algoritması ile Petri ağının minimum T-değişmezleri[2] bulunur.

$\mathcal{T}$=Algoritma-2[G]

1. $A' = [I \vdots A^T]$ şeklinde oluşturulur.
2. j=$|T|$+1
3. j. sütundaki elemanlardan 0'dan farklı olup toplamları 0'ı veren çiftler bulunur.
4. Bu elemanların bulundukları satırlar toplanır.
5. Her bir satır çiftinin toplamları A' matrisinin altına eklenir.

[2]Bir Petri ağının sonsuz tane T-değişmezi olabilir. Minimum T-değişmezleri ise, elemanları yine doğal sayı olan, ağın diğer T-değişmezlerinin doğal katsayılar ile doğrusal kombinasyonu olarak yazılamayan T- değişmezleridir.

6. Toplam için kullanılan tüm satırlar A' matrisinden silinir.

7. Eğer $j = |P| + |T|$ ise, 8. adıma geçilir; aksi durumda $j = j + 1$ yapılarak 3'e dönülür.

8. A' matrisinde $|P| + 1$. sütundan itibaren 0 olan satırların ilk n elemanı T-değişmezinin transpozunu verir.

Yukarıdaki algoritma için geliştirilen Tinv.m adlı MATLAB programı Ek-B'de verilmiştir.

Örneğin, aşağıdaki gibi bir çakışım matrisine sahip bir Petri ağı için, rank$(A) \neq 3$ olduğundan, ağın $\mathcal{O}$' dan farklı T-değişmezi bulunmaktadır (bkz. Bölüm 3.3.1).

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

Bu Petri ağı için Tinv.m programının basamak basamak çalışması, aşağıdaki gibidir;

$$A^T = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & 0 \end{bmatrix} \qquad A' = \begin{bmatrix} 1 & 0 & 0 & -1 & 1 & 1 \\ 0 & 1 & 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 1 & -1 & 0 \end{bmatrix}$$

4. sütunun sıfırlanması

$$\begin{matrix} 1 \\ 2 \\ 3 \\ 1+2 \\ 1+3 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & -1 & 1 & 1 \\ 0 & 1 & 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

1., 2. ve 3. satırlar toplam için kullanıldığından silinmelidir.

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

5. sütunun sıfırlanması

Bu sütun bir önceki adımda yapılan işlemlerle kendiliğinden sıfırlandı.

6. sütunun sıfırlanması

Bu sütunun sıfırlanması mümkün değil.

Bu durumda Petri ağının minimum T-Değişmezi $[1\ 1\ 0]^T$ şeklinde bulunmaktadır. yani bir M işaretleme vektöründen t_1 ve t_2 geçişlerinin birer kez kullanıldığı bir geçiş dizisinin ($g = t_1t_2$ veya $g = t_2t_1$) ateşlenebilmesi durumunda yine M işaretleme vektörüne dönülür.

3.4 Çakışım Matrisi ile Petri Ağı Analizi

G ile ifade edilen bir Petri ağı için çakışım matrisi ve T değişmezi bulunduğunda ağın birçok özelliği hakkında bilgi edinilmektedir. [12. 13. 4]'den faydalanılarak, bunlardan bazıları, bu bölümde özetlenecektir.

Bir Petri ağı için $AW = \mathcal{O}$ denkleminin tek çözümü $W = \mathcal{O}$ ise bu ağ tersine dönüşebilir değildir.

Bir Petri ağı için $AW = \mathcal{O}$ denkleminin $W = \mathcal{O}$ dan farklı en az bir çözümü varsa bu ağ en az bir başlangıç durumu için kısmi tersine dönüşebilirdir [13] ve ağın ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesi vardır.

Bir Petri ağında T değişmezi analizi yapılarak, ağın özellikleri hakkında edinilen bilgilere aşağıdakiler de eklenebilir.

Lemma 3.1: Bir Petri ağı için $AW = \mathcal{O}$ denkleminin tek çözümü $W = \mathcal{O}$ ise bu ağ kısmi tersine dönüşebilir değildir.

İspat: Eğer bir Petri ağının tüm T-Değişmezlerinde aynı geçişe karşılık gelen

eleman 0 ise, bu geçiş ateşlendiğinde hiçbir şekilde başlangıç durumuna dönmek mümkün değildir [4]. Buna göre bir Petri ağının yalnız bir T-Değişmezi varsa ve bu da $\mathcal{O}$'a eşitse hiçbir geçişin ateşlenmesi ile başlangıç durumuna dönmek mümkün değildir.

Lemma 3.2: Eğer bir Petri ağında, $g = t_{i_1}t_{i_2}....t_{i_n}$, $(t_{i_1}, t_{i_2}....t_{i_n} \in T)$ geçiş dizisi ağın $\mathcal{O}$'dan farklı bir T-değişmezine karşılık gelen bir geçiş dizisi ve $g_1 = t_{i_1}t_{i_2}..t_{i_k}$, $(n > k)$ (g_1, g geçiş dizisinin bir bölümü) şeklinde tanımlanmak üzere $\rho(m_0, g) = m_0$ ve $\rho(m_0, g_1) = M \neq m_0$ ateşlemeleri mümkünse bu Petri ağı kısmi tersine dönüşebilirdir.

İspat: T-değişmezinin tanımına göre, Petri ağının bir T-değişmezine karşılık gelen herhangi bir geçiş dizisinin bir işaretleme vektöründen ateşlenebilmesi durumunda, yeniden aynı işaretleme vektörüne dönülecektir. Öylevse bu özellikteki bir geçiş dizisi m_0'dan ateşlenebiliyorsa m_0'a geri dönülecektir. Bu ateşlemelerin herhangi bir yerinde m_0'dan farklı en az bir işaretleme vektörü oluşuyorsa, m_0'dan farklı en az bir işaretleme vektörü de m_0'a dönebildiğinden Petri ağı kısmi tersine dönüşebilirdir.

Örnek 3.2:

Şekil 3.4'deki Petri ağının T-değişmezi $[1\ 1\ 1\ 1\ 1\ 1\ 1]^T$ şeklinde bulunur (bkz. Bölüm 3.3.2). Bu ağ sınırlıdır. Bu Petri ağı için T-değişmezi $\mathcal{O}$'dan farklı olduğundan, ağ en az bir başlangıç vektörü için kısmı tersine dönüşebilirdir. Ağın T-değişmezi $[1\ 1\ 1\ 1\ 1\ 1\ 1]^T$ olduğundan "$t_1, t_2, t_3, t_4, t_5, t_6$" geçişlerinin herbirinin birer kez kullanıldığı bir geçiş dizisi bir işaretleme vektöründen ateşlenebildiğinde, ateşlemenin başladığı vektöre geri dönülür. Ağ için oluşturulan kapsayabilirlik ağacında böyle bir geçiş dizisinin m_0'dan ateşlenebildiği görülmektedir. Şekil 3.5'e göre $\rho(m_0, t_1) = [2\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1]^T = M_1$, $\rho(M_1, t_2) = [2\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1]^T = M_2, \rho(M_2, t_3) = [2\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1]^T = M_3$, $\rho(M_3, t_4) = [2\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0]^T = M_4, \rho(M_4, t_5) = [2\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0]^T = M_5$, $\rho(M_5, t_6) = [3\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]^T = M_6 = m_0$ şeklindedir. Bu durumda başlangıç işaretleme vektörü, $[3\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]^T$ iken bu Petri ağı kısmi tersine dönüşebilirdir.

4. TERSİNE DÖNÜŞEBİLİRLİK

Sınırsız Petri ağlarının ulaşılabilirlik kümesinde sonsuz tane eleman bulunur. [11]'de geliştirilen Bounded-Set algoritması ile, daha önceden belirlenmiş bir K sınır vektörü kullanılarak sınırsız bir Petri ağının ulaşılabilirlik kümesinin sonlu sayıda elemandan oluşan ve R_B ile gösterilen bir alt kümesi elde edilmektedir. Bu kümenin elemanları için $K = [k_1\ k_2....k_{|P|}]^T$ olmak üzere,

$$M(p) \leq K(p),\ \forall M \in R_B(G, m_0) \text{ ve } \forall p \in P$$

şartı sağlanmaktadır.

Bu bölümde sunulacak iki yöntem ile kısmi tersine dönüşebilir Petri ağları için K sınır vektörleri bulunur. Bu sınır vektörleri yukarıda bahsedilen algoritmaya girdi olarak verildiğinde, ilgili Petri ağının ulaşılabilirlik kümesinin kısmi tersine dönüşebilir bir alt kümesi elde edilmektedir. Yine [11]'de geliştirilen Reversible-Set algoritması ile de bu kümenin tersine dönüşebilir kısmı, diğer bir deyişle Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesi elde edilir. Sunulan yöntemlerle elde edilen sınır vektörleri Petri ağının tersine dönüşebilirliğini garanti eder.

4.1 Yöntem 1

Yöntem 1, sınırsız temel bir Petri ağının kısmi tersine dönüşebilirliğini test etmekte, ağ kısmi tersine dönüşebilir ise ağın tersine dönüşebilirliğini garanti eden bir sınır vektörü önermektedir.

Bu yöntem oluşturulurken temel alınan özellikler ve lemmalar aşağıda verilmiştir.

Özellik 4.1: Temel bir Petri ağında, ağın yapısındaki okların hepsinin ağırlığı 1 olduğundan, tek bir geçiş ateşlenmesiyle herhangi bir yerde en fazla 1 belirti değişimi meydana gelmektedir. Buna göre temel bir Petri ağında kendisinden tek bir geçiş ateşlenmesi ile m_0'a ulaşan $R(G, m_0)$ elemanı tüm işaretleme

vektörlerinin oluşturduğu küme,

$$R' = \{M \in R(G, m_0) \mid \rho(M, t) = m_0,\ t \in T;\} \tag{4.1}$$

şeklinde tanımlanır ve temel Petri ağları sözkonusu olduğundan $\forall M \in R'$ için $M(p) \pm a = m_0(p)$, $a \in \{0, 1\}$, $\forall p \in P$ şeklindedir.

Lemma 4.1: Eğer temel bir Petri ağında R' kümesinde $M \neq m_0$ şeklinde bir işaretleme vektörü bulunmuyorsa, Petri ağı kısmi tersine dönüşebilir değildir (Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesi yoktur).

İspat: Bir Petri ağında herhangi bir $\bar{M} \in R(G, m_0) \backslash R'$ için, $\rho(\bar{M}, g) = m_0$ mümkünse; bunu sağlayan en küçük geçiş dizisi $g = t_{i_1} t_{i_2} t_{i_3} .... t_{i_k}$. ($t_{i_1}$, $t_{i_2}, t_{i_3} .... t_{i_k} \in T$) şeklinde ve $g_1 = t_{i_1} t_{i_2} t_{i_3} .... t_{i_{k-1}}$ şeklinde tanımlanmak üzere. $\rho(\bar{M}, g_1) = M$, $M \neq m_0$ ve $\rho(M, t_{i_k}) = m_0$ şeklinde elde edilmektedir. R' kümesinin tanımı gereği burada $M \in R'$ olmalıdır. Dolayısıyla Petri ağında R' kümesinde $M \neq m_0$ şeklinde bir işaretleme vektörü yoksa Petri ağı kısmi tersine dönüşebilir değildir.

Sınırsız Petri ağlarında $R(G, m_0)$ sonsuz sayıda eleman içerir. Bundan dolayı, $R(G, m_0)$ kullanılarak tanımlanan R' kümesini (bkz. Eşitlik 4.1) elde etmek amacıyla, kapsayabilirlik ağacından faydalanılarak, sınırlı sayıda eleman içeren ve R' kümesini kapsadığı bilinen bir $\tilde{R}$ kümesi oluşturulur (bkz. Algoritma 4).

Özellik 4.2: $\tilde{R}$, m_0'dan başlayarak yapılan ateşlemelerle elde edilen sınırlı sayıdaki işaretleme vektöründen oluşan ve

$$R(G, m_0) \supset \tilde{R} \supset R' \tag{4.2}$$

şartını sağlayan bir küme olarak tanımlansın. $M \in R'$, $M \neq m_0$ şeklinde en az bir işaretleme vektörü varsa hem $R(G, m_0)$ hem de $\tilde{R}$ kısmi tersine dönüşebilirdir, $\tilde{R}$ kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt

kümesini içerir.

Kapsayabilirlik ağacındaki vektörler ile $R(G, m_0)$ elemanı işaretleme vektörleri arasındaki ilişki Özellik 4.3 ve Özellik 4.4'de verilmiştir.

Özellik 4.3: Eğer $\tilde{M} >_d M$ ve $\tilde{M} = \rho(M, g)$, $\delta = \tilde{M} - M$ ise, $\rho(\tilde{M} + k\delta, g) = \tilde{M} + (k+1)\delta$. $(k \in D)$ olarak elde edilir [16]. Buna göre bir Petri ağında bir işaretleme vektöründen yapılan ateşlemelerle kendisine baskın bir işaretleme vektörü elde ediliyorsa, yeni elde edilen işaretleme vektöründen yapılan ateşlemelerle bu vektöre baskın yeni vektörler de elde edilir. Petri ağı sınırsızdır.

Eğer $\tilde{M} >_d M$ ve $\tilde{M} = \rho(M, g)$ ise kapsayabilirlik ağacında bu. $\tilde{M}$ vektörünün $\tilde{M}(p) > M(p)$ şartını sağlayan yerlerinde w sembolü kullanılarak ifade edilir ve kapsayabilirlik ağacındaki w içeren bir vektör ulaşılabilirlik kümesinde birden fazla işaretleme vektörünü ifade eder.

Örneğin M, $\tilde{M} \in R(G, m_0)$ iken $M = [1\ 1\ 0]^T$, $\tilde{M} = [2\ 1\ 0]^T$ ve $\rho(M, g) = \tilde{M}$ olsun. $\rho(\tilde{M}, g) = \tilde{M}_1 = [3\ 1\ 0]^T$, $\rho(\tilde{M}_1, g) = \tilde{M}_2 = [4\ 1\ 0]^T$, ... şeklindedir $(\tilde{M}_i \in R(G. m_0),\ i \in D)$. Ağın kapsayabilirlik ağacında $[w\ 1\ 0]$ şeklinde bir vektör bulunur ve bu vektör ulaşılabilirlik kümesindeki $\tilde{M}$ ve tüm $\tilde{M}_i$ işaretleme vektörlerini ifade eder.

Özellik 4.4: $\tilde{M} >_d M$ ise $E(G, \tilde{M}) \supset E(G, M)$ şeklinde elde edilir.

Yöntem 1 için aşağıda verilen algoritmada, öncelikle ele alınan Petri ağının rank(A) değerine bakılır ve ağın $\mathcal{O}$'dan farklı bir T-değişmezi yoksa, ağ kısmi tersine dönüşebilir olmadığından (bkz. Lemma 3.1) algoritma sonlandırılır. Eğer Petri ağının $\mathcal{O}$'dan farklı en az bir T-değişmezi varsa; öncelikle Algoritma-1 ile (bkz. Bölüm 3.1) φ, φ_1, φ_2 kümeleri elde edilir. m_0'dan başlanarak yapılan ateşlemelerle φ_2 kümesindeki vektörlerin ulaşılabilirlik kümesinde en az belirti taşıyan hallerinin bulunduğu R_0 kümesini oluşturan Algoritma-3 çağrılır. R_0 kümesi elde edildikten sonra, ateşlemelere kalınan yerden devam edilmesi halinde yeni elde edilen herhangi bir M işaretleme

vektörü, en az bir sınırsız yerde daha fazla belirti taşımak kaydıyla daha önce oluşan en az bir işaretleme vektörüne baskındır. Daha sonra Algoritma-4 çağrılır ve ateşlemelere kalınan yerden devam edilip. $i = 1$'den başlanarak $R_i \cap R' = \emptyset$ şartını sağlayan bir R_i kümesi bulunana kadar R_i kümeleri oluşturulur. R_i; R_{i-1} kümesindeki işaretleme vektörlerinin, en az bir sınırsız yerinde daha fazla belirti taşıyan hallerinin bulunduğu kümedir (Her bir R_i kümesi R_{i-1} kümesinde son kalınan yerden ateşlemelere başlanarak elde edilen işaretleme vektörleri kümesidir). Buna göre $R_i \cap R' = \emptyset$ ise ($\forall M \in R_i$ ve en az bir $p \in \tilde{P}$ için $M(p) \geq m_0(p) + 2$ ise); $j > i$ iken $R_j \cap R' = \emptyset$ olacağından ($\forall M \in R'$, $M(p) \pm a = m_0(p)$, $a \in \{0, 1\}$, $\forall p \in P$), $R_i \cap R' = \emptyset$ şartını sağlayan bir R_i bulunduğunda, o ana kadar elde edilen tüm işaretleme vektörleri ile $\tilde{R}$ kümesi ($\tilde{R} \supset R'$) oluşturulur (bkz. Özellik 4.2). Daha sonra Eşitlik 4.1'den faydalanılarak Petri ağının R' kümesi oluşturulur. $M \in R'$, $M \neq m_0$ şeklinde bir M işaretleme vektörü varsa Petri ağı kısmi tersine dönüşebilirdir ve ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü 6. basamakta önerilen yaklaşımlardan biriyle bulunur. Aksi takdirde Petri ağı kısmi tersine dönüşebilir değildir (bkz. Lemma 4.1). Algoritma bir sınır vektörü bulmadan sonlanır.

[K]=Yöntem1 [G] Algoritması

1. Eğer rank(A)$\neq |T|$ ise, 2. adıma geçilir. Aksi durumda algoritma sonlanır.
2. **Algoritma-1** (bkz. Bölüm 3.1) çağrılır, φ, φ_1, φ_2 kümeleri elde edilir.
3. **Algoritma-3** çağrılır, R_0 kümesi elde edilir.
4. **Algoritma-4** çağrılır, $\tilde{R}$ kümesi elde edilir.
5. Petri ağı için R' kümesi

$$R' = \{M \in \tilde{R} \mid \rho(M, t) = m_0,\ t \in T\} \tag{4.3}$$

şeklinde oluşturulur. $M \neq m_0$ şeklinde en az $M \in R'$ işaretleme vektörü varsa 6. adıma geçilir. Aksi takdirde Petri ağı kısmi tersine dönüşebilir değildir, algoritma sonlanır.

6. K sınır vektörü aşağıdaki yaklaşımlardan biriyle elde edilir.

$$a-)K(p) = \max_{M \in \tilde{R}}(M(p)),\ \forall p \in P \tag{4.4}$$

$b-)$**Algoritma-5** çağrılır.

Yöntem 1 algoritmasında K sınır vektörünün bulunması için kullanılan yaklaşımlardan Algoritma-5'de, sınır vektörü sadece $\tilde{R}$ kümesindeki R' işaretleme vektörlerinin oluşması ve m_0'a dönmesi esnasında oluşan işaretleme vektörlerini kapsayacak şekilde belirlenmektedir. Oysa ki Eşitlik 4.4 ile $\tilde{R}$ kümesindeki tüm işaretleme vektörlerini kapsayan bir sınır vektörü elde edilmektedir. Dolayısıyla şöyle bir genelleme yapılabilir: Algoritma-5 ile elde edilen sınır vektörü K_1 ve Eşitlik 4.4 ile elde edilen sınır vektörü K_2 iken $K_1(p) \leq K_2(p),\ \forall p \in P$ şeklindedir.

Aşağıda verilen Algoritma-3 ve Algoritma-4'de geçen, $\mathcal{S}$, bir sonraki R_i kümesi oluşturulurken ateşlemelerin başlayacağı işaretleme vektörlerine; $\mathcal{E}$, daha önce oluşmuş işaretleme vektörlerine; $\mathcal{D}$, çıkmaz işaretleme vektörlerine verilen etikettir. Bu algoritmaların inşaasında kapsayabilirlik ağacı algoritmasından faydalanılmıştır.

$[SM, R_0]$=**Algoritma-3** $[G, \varphi]$

1. m_0 ağacın en başına yerleştirilir, m_0'dan ateşlenebilecek tüm geçişler ateşlenerek yeni işaretleme vektörleri oluşturulur.

2. İşaretleme vektörleri içindeki $\mathcal{E}, \mathcal{D}$ veya $\mathcal{S}$ etiketli olmayan yeni işaretleme vektörleri sırayla seçilerek herbiri için şu işlem yapılır: (seçilen işaretleme vektörü $\tilde{M}$ olarak gösterilsin)

   (a) Eğer m_0'dan $\tilde{M}$ oluşana kadar olan yolda $\tilde{M} >_d M$ ve $E(G, M) =$

$E(G, \tilde{M})$ şeklinde bir M işaretleme vektörü varsa, bu vektöre $\mathcal{S}$ etiketi verilir, 2. adıma dönülür.

(b) Eğer daha önce $\tilde{M}$ işaretleme vektörü elde edilmiş ise, bu vektöre $\mathcal{E}$ etiketi verilir, 2. adıma dönülür.

(c) Eğer $\tilde{M}$ işaretleme vektöründen hiçbir geçiş ateşlenmiyorsa, bu vektöre $\mathcal{D}$ etiketi verilir, 2. adıma dönülür.

(d) Diğer durumlarda $\tilde{M}$ işaretleme vektöründen ateşlenebilecek geçişler ateşlenerek, yeni işaretleme vektörleri elde edilir, 2. adıma dönülür.

3. Bu adıma gelininceye kadar oluşan işaretleme vektörleri içinde, φ_1 kümesindeki tüm işaretleme vektörleri ve herbir φ_2 elemanına karşılık en az bir işaretleme vektörü bulunuyorsa, $\mathcal{S}$ etiketli işaretleme vektörleri ile SM kümesi oluşturulur, oluşan tüm işaretleme vektörleri ile R_0 kümesi oluşturulur ve algoritma sonlanır. Aksi takdirde, $\mathcal{S}$ etiketli işaretleme vektörlerinin etiketleri kaldırılır, bu vektörlerden ateşlenebilecek tüm geçişler ateşlenerek yeni işaretleme vektörleri oluşturulur, 2. adıma dönülür.

Algoritma-3 ile belirlenen R_0 işaretleme vektörleri kümesi tüm φ_1 işaretleme vektörlerinin ve Petri ağının kapsayabilirlik ağacındaki tüm φ_2 elemanlarının ulaşılabilirlik kümesinde en az belirti taşıyan hallerinin bulunduğu kümedir.

$[\tilde{R}]$=**Algoritma-4** $[G, SM]$

1. $i = 1$

2. $SM_0 = SM$ yapılır. SM_0 kümesindeki işaretleme vektörleri ateşlemelerin başlayacağı işaretleme vektörleridir, $\mathcal{S}$ ile etiketlenir.

3. SM_{i-1} kümesindeki işaretleme vektörlerinden ateşlenebilecek tüm geçişler ateşlenerek, yeni işaretleme vektörleri oluşturulur.

4. Yeni işaretleme vektörleri içindeki $\mathcal{E}, \mathcal{D}$ veya $\mathcal{S}$ etiketli olmayan işaretleme

vektörleri sırayla seçilerek şu işlemler yapılır: (seçilen işaretleme vektörü $\tilde{M}$ olarak gösterilsin)

(a) Eğer $\tilde{M} >_d M$ ve $E(G, M) = E(G, \tilde{M})$ şeklinde SM_{i-1} elemanı bir M işaretleme vektörü varsa, bu vektöre $\mathcal{S}$ etiketi verilir. 4.adıma dönülür.

(b) Eğer $\tilde{M}$ daha önce elde edildiyse, bu vektöre $\mathcal{E}$ etiketi verilir. 4.adıma dönülür.

(c) Eğer $\tilde{M}$ işaretleme vektöründen hiçbir geçiş ateşlenmiyorsa, bu vektöre $\mathcal{D}$ etiketi verilir, 4. adıma dönülür.

(d) Diğer durumlarda $\tilde{M}$ işaretleme vektöründen ateşlenebilecek geçişler ateşlenerek, yeni işaretleme vektörleri elde edilir. 4.adıma dönülür.

5. SM_{i-1} kümesindeki işaretleme vektörlerinden yapılan ateşlemelerle elde edilen $\mathcal{E}$ etiketli olmayan tüm işaretleme vektörleri ile R_i kümesi oluşturulur.

6. $\forall M \in R_i$ ve en az bir $p \in \tilde{P}$ için $M(p) \geq m_0(p)+2$ sağlanıyorsa (bu şartın sağlanması $R_i \cap R' = \emptyset$ anlamına gelir) $\tilde{R} = \bigcup_{j=0}^{i-1} R_j$ şeklinde oluşturulur. algoritma sonlanır. Aksi takdirde, $i = i + 1$ yapılır ve 4.adımda $\mathcal{S}$ ile etiketlenen işaretleme vektörleri SM_{i-1} kümesi içine atılarak 3.adıma dönülür.

Algoritma-4 ile elde edilen $\tilde{R}$, m_0'dan başlayarak yapılan ateşlemelerle elde edilen işaretleme vektörlerinden oluşan, kapsayabilirlik ağacındaki her bir vektörün ulaşılabilirlik kümesinde ifade ettiği en az bir işaretleme vektörünü içeren ve Petri ağının R' kümesini kapsayan bir işaretleme vektörleri kümesidir (bkz. Özellik 4.2).

[K]=Algoritma-5$[G, \tilde{R}]$

1. $PR = \emptyset$, $K_{buf} = \emptyset$

2. Herbir $M \in R'$ işaretleme vektörü sırayla seçilerek herbiri için şu işlem yapılır:

   2.1 $\rho(m_0, g) = M$ ateşlemesinde elde edilen tüm işaretleme vektörleri PR kümesi içine atılır.

   $$K_x(p) = \max_{M' \in PR}(M'(p)),\ \forall p \in P \tag{4.5}$$

   şeklinde belirlenir, K_{buf}=$K_{buf} \cup K_x$, $PR = \emptyset$ yapılır, 2.adıma dönülür.

3. K vektörü,

   $$K(p) = \max_{K_x \in K_{buf}}(K_x(p)),\ \forall p \in P \tag{4.6}$$

   şeklinde belirlenir.

Algoritma-5 ile bulunan K, tüm R' elemanlarının oluşması ve yeniden m_0'a dönmeleri esnasında oluşan tüm işaretleme vektörlerini kapsayan bir sınır vektörüdür.

Yöntem 1 algoritması için geliştirilen Yöntem1.m adlı Matlab programı Ek-C'de verilmiştir. Bu programda K sınır vektörünün belirlenmesi aşamasında Algoritma-5 kullanılmaktadır.

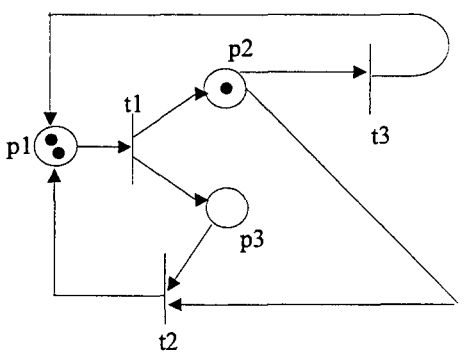
Teorem 4.1: Yöntem 1 Algoritması kullanılarak elde edilen K sınır vektörü (bkz. Eşitlik 4.4) Petri ağının tersine dönüşebilirliğini garanti eden bir sınır vektörüdür.

İspat: Yöntem 1 algoritmasında elde edilen $\tilde{R}$ kümesi ile Petri ağının R' kümesi arasında $\tilde{R} \supset R'$ şeklinde bir ilişki sözkonusudur. R' elemanı bir M işaretleme vektörünün $M \neq m_0$ şartını sağlaması durumunda $\tilde{R}$ kısmi tersine dönüşebilirdir ve K sınır vektörü belirlenir (bkz. Özellik 4.2). Belirlenen K, $\tilde{R}$ kümesindeki tüm işaretleme vektörlerini, dolayısıyla Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm vektörleri kapsadığından (bkz. Eşitlik 4.4) Petri ağının tersine dönüşebilirliğini garanti etmektedir.

Teorem 4.2: Yöntem 1'de Algoritma-5 kullanılarak elde edilen K sınır vektörü Petri ağının tersine dönüşebilirliğini garanti eden bir sınır vektörüdür.

İspat: Bir Petri ağının kısmi tersine dönüşebilir olduğu durumda, en az bir $M \in R'$ için, $M \neq m_0$ şeklindedir ve R' kümesindeki tüm işaretleme vektörleri $\rho(M,t) = m_0$ eşitliğini sağladığından, hepsi m_0'a ulaşmaktadır (bkz. Eşitlik 4.1). Algoritma-5 ile elde edilen K sınır vektörü, her bir $M \in R'$ için , bir $\rho(m_0, g) = M$ ve $\rho(M,t) = m_0$ ateşlemesi esnasında oluşan her işaretleme vektörlerini kapsamaktadır. Bu işaretleme vektörlerinin herbiri m_0'a ulaşabilir. Bu durumda K, Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsar ve dolayısıyla Petri ağının tersine dönüşebilirliğini garanti eder.

Örneğin, Şekil 4.1'deki Petri ağı [4] için rank$(A) \neq |T| = 3$ olduğundan bu ağın $\mathcal{O}$'dan farklı bir T-değişmezi vardır, ağ en az bir başlangıç vektörü için kısmi tersine dönüşebilirdir.



Şekil 4.1: Örnek bir Petri ağı

Bu örnek ağ için Yöntem1.m programı çalıştırıldığında aşağıdaki veriler elde edilir;

- $\varphi(G, m_0) = \{[2\ 1\ 0]^T,\ [1\ 2\ 1]^T,\ [3\ 0\ 0]^T,\ [0\ 3\ 2]^T,\ [2\ 1\ w]^T,\ [1\ 2\ w]^T,\ [3\ 0\ w]^T,\ [0\ 3\ w]^T\}$
- $R_0 = \{[2\ 1\ 0]^T,\ [1\ 2\ 1]^T,\ [3\ 0\ 0]^T,\ [0\ 3\ 2]^T,\ [2\ 1\ 1]^T,\ [1\ 2\ 2]^T,\ [3\ 0\ 1]^T\}$

- $R_1 = \{[0\ 3\ 3]^T,\ [1\ 2\ 3]^T,\ [3\ 0\ 2]^T,\ [2\ 1\ 2]^T\}$

Burada, $\forall M \in R_1$ ve en az bir sınırsız p yeri için $M(p) \geq m_0(p) + 2$ sağlandığından,

$$R_1 \cap R' = \emptyset \text{ ve } \tilde{R} = R_0$$

olarak belirlenir. Bunun üzerine Petri ağının R' kümesi Eşitlik 4.3'den yararlanılarak, $R' = \{[1\ 2\ 1]^T\}$ olarak elde edilmektedir. Petri ağı kısmi tersine dönüşebilirdir. $\tilde{R}$ kümesi içinde m_0'dan $[1\ 2\ 1]^T$ işaretleme vektörü ve yine m_0 oluşana kadarki yol; $\rho([2\ 1\ 0]^T, t_1) = [1\ 2\ 1]^T$, $\rho([1\ 2\ 1]^T, t_2) = [2\ 1\ 0]^T$ şeklinde olduğundan, bu Petri ağının kısmi tersine dönüşebilirliğini garanti eden K sınır vektörü, Algoritma-5'e göre $K = [2\ 2\ 1]^T$ şeklinde elde edilmektedir. Yöntem1.m programının çıktı dosyasında $\tilde{R}$, R' ve K bilgilerine yer verilmektedir (bkz. Şekil 4.2).

```
R                R'    K
210   122        121   221
121   300
032
211
301
```

Şekil 4.2: Şekil 4.1 için Yöntem1.m programının çıktı dosyası

Elde edilen K sınır vektörü kullanılarak, sırasıyla Bounded-Set ve Reversible-Set algoritmaları [11] çalıştırıldığında Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesi ($R_r = \{[2\ 1\ 0]^T,\ [1\ 2\ 1]^T\}$) oluşturulur.

Bu Petri ağı için, Yöntem 1 algoritmasında verilen Eşitlik 4.4 ile elde edilen sınır vektörü $[3\ 3\ 2]^T$ ve bu sınır vektörü kullanılarak, sırasıyla Bounded-Set ve Reversible-Set algoritmaları çalıştırıldığında elde edilen tersine dönüşebilir küme $R_r = \{[2\ 1\ 0]^T,\ [1\ 2\ 1]^T, [0\ 3\ 2]^T\}$ şeklindedir.

4.2 Yöntem 2

Bu bölümde kısmi tersine dönüşebilir temel bir Petri ağının bu özelliğini garanti eden bir K sınır vektörü Algoritma-2 (bkz. Bölüm 3.3.2) ile bulunan T-değişmezleri kullanılarak elde edilecektir.

Yöntem 2 için aşağıda verilen algoritmada, öncelikle Petri ağının rank(A) değerine bakılır ve ağın $\mathcal{O}$'dan farklı bir T-değişmezi yoksa, ağın kısmi tersine dönüşebilir olmadığı sonucuna ulaşılır (bkz. Lemma 3.1). Eğer Petri ağının $\mathcal{O}$'dan farklı en az bir T-değişmezi varsa Algoritma-2 (bkz. Bölüm 3.3.2) ile Petri ağının minimum T-değişmezleri bulunur ve bunlarla $\mathcal{T}$ kümesi oluşturulur. Algoritma-6 ile bu T-değişmezilerine karşılık gelen geçiş dizilerinden m_0'dan ateşlenebilen olup olmadığı kontrol edilir. Eğer böyle bir geçiş dizisi varsa Petri ağı kısmi tersine dönüşebilirdir ve bulunan geçiş dizilerinin ateşlenmesi esnasında oluşacak tüm işaretleme vektörlerini kapsayacak bir sınır vektörü, Petri ağının tersine dönüşebilirliğini garanti eden K sınır vektörü olarak seçilir. Eğer Algoritma-2 ile elde edilen T-değişmezlerine karşılık gelen geçiş dizilerinden m_0'dan ateşlenebilen yoksa, algoritma bir sınır vektörü bulmadan sonlanır. Ancak bu, ağın kısmi tersine dönüşebilir olmadığı anlamına gelmez. Çünkü Petri ağının $\mathcal{O}$'dan farklı en az bir T-değişmezi varsa, sonsuz tane T-değişmezi vardır. Yöntem 2'de ise sadece Algoritma-2 ile bulunan T-değişmezlerinin m_0'dan ateşlenebilirliği kontrol edilmektedir.

Yöntem 2 Algoritması

1. Eğer rank(A)$\neq$ $|T|$ ise, 2. adıma geçilir. Aksi durumda algoritma sonlanır.
2. **Algoritma-2** çağrılır, bulunan T-değişmezleri ile $\mathcal{T}$ kümesi oluşturulur.
3. **Algoritma-6** çağrılır, K_{buf} kümesi oluşturulur.
4. Sınır vektörü

$$K(p) = \max_{K_x \in K_{buf}} (K_x(p)),\ \forall p \in P \tag{4.7}$$

şeklinde belirlenir.

Algoritma-6'da kullanılan ve elde edilen her bir M işaretleme vektörü için oluşturulan $GY(M)$, m_0'dan M işaretleme vektörü oluşana kadar ateşlenen geçiş dizisinin ateşleme sayısı vektörünü göstermektedir. Örneğin, $T = \{t_1,\ t_2,\ t_3,\ t_4,\ t_5\}$ iken $\rho(m_0, t_3t_2t_5t_2) = M$ şeklinde oluşan bir M işaretleme vektörü için $GY(M) = [0\ 2\ 1\ 0\ 1]^T$ olarak belirlenir. Petri ağının ulaşılabilirlik kümesindeki bir M işaretleme vektörü m_0'dan ateşlenen farklı geçiş dizileri ile farklı yollardan (geçiş ve işaretleme vektörlerinden oluşan bir dizi) birden fazla kere oluşmuş olabilir. Algoritma-6, m_0'dan farklı yollardan oluşan işaretleme vektörlerini, farklı vektörler olarak kabul eder ve herbiri için ayrı $GY(.)$ oluşturur.

Algoritma-6 ile T_j vektörleri(T_j: $\mathcal{T}$ kümesindeki j.T-değişmezi, $j \in \{1, 2, ..|T|\}$) sırayla seçilmekte ve herbiri için şu işlemler yapılmaktadır:

m_0'dan ateşlemelere başlanarak yeni işaretleme vektörleri oluşturulmaktadır. Oluşan her M işaretleme vektörü için $GY(.)$ vektörü belirlenmektedir. $T_j - GY(M) = \alpha$ iken,

- $\alpha(i) < 0$ en az bir $i \in \{1, 2...|T|\}$ ise;
  i ninci geçiş, g_1 geçiş dizisinde T_j'de ifade edilenden daha fazla ateşlendiğinden, T_j değişmezinin ifade ettiği bir geçiş dizisinin m_0'dan ateşlenmesi ile m_0'a dönülemez. Bu durumda algoritma M işaretleme vektörünü $\mathcal{X}$ olarak etiketlemekte ve bu vektörden başka geçiş ateşlememektedir.
- $\alpha(i) \geq 0$ $i \in \{1, 2...|T|\}$ ve $\alpha \neq \mathcal{O}$ ise;
  g_1 geçiş dizisinde tüm geçişler T_j'de ifade edildiğinden daha az veya eşit sayıda ateşlendiğinden, $\rho(M, g_2) = m_0$ şeklinde bir g_2 geçiş dizisi bulunma ihtimali korunmaktadır. Bu durumda algoritma M işaretleme vektörünü $\mathcal{N}$ ile etiketlenerek ve bu

vektörden ateşlemelere devam ederek g_2 geçiş dizisinin varlığını kontrol etmektedir.

- $\alpha = \mathcal{O}$ ise $\rho(m_0, g_1) = M = m_0$ şeklindedir. m_0'dan yeniden m_0 (M) elde edilirken $M' \neq m_0$ şeklinde en az bir işaretleme vektörü elde ediliyorsa, Petri ağı kısmi tersine dönüşebilirdir. m_0'dan g_1 geçiş dizisinin ateşlenmesi esnasında oluşan tüm işaretleme vektörlerini (m_0'dan M' işaretleme vektörüne ulaşılırken ve M' dan yeniden m_0'a ulaşılırken oluşan tüm işaretleme vektörlerini) kapsayan bir sınır vektörü (K_x) ağın kısmi tersine dönüşebilirliğini garanti eder. Algoritma-6. $\rho(m_0, g_1) = M = m_0$ $(g = g_1)$ şeklinde elde edilen tüm M işaretleme vektörleri için, yukarıda bahsedilen şekilde K_x sınır vektörleri belirlemekte ve tüm K_x sınır vektörlerinin kapsadığı işaretleme vektörlerini kapsayacak en küçük K sınır vektörünü ağın kısmi tersine dönüşebilirliğini garanti eden sınır vektörü olarak seçmektedir.

Yine bu algoritmada geçen $\mathcal{X}$ kendisinden ateşleme yapılmayacak işaretleme vektörlerine, $\mathcal{N}$ kendisinden yapılan ateşlemelere devam edilecek işaretleme vektörlerine verilen etikettir.

$[K_{buf}]$=**Algoritma-6** $[G, \mathcal{T}]$

1. $s = |\mathcal{T}|$, $K_x = K_{buf} = \emptyset$, $j = 1$ şeklinde belirlenir. T_j, $\mathcal{T}$ kümesinin j.elemanıdır.[3]

2. m_0, $\mathcal{N}$ olarak etiketlenir, ağacın üstüne yerleştirilir.

3. $\mathcal{N}$ ile etiketli işaretleme vektörleri sırayla seçilerek herbiri için şu işlemler yapılır: (seçilen işaretleme vektörü M olsun)

   3.1 M vektörü için $GY(M)$ vektörü oluşturulur.

[3]Örneğin bir Petri ağının Algoritma-5 ile bulunan T-değişmezleri, $[0\ 1\ 1]^T$, $[1\ 1\ 0]^T$ şeklinde ise, $\mathcal{T} = \{[0\ 1\ 1]^T, [1\ 1\ 0]^T\}$; $T_1 = [0\ 1\ 1]^T$, $T_2 = [1\ 1\ 0]^T$.

3.2 $T_j - GY(M)=\alpha$ oluşturulur.

(a) Eğer $\alpha(i) < 0$ en az bir $i \in \{1, 2....|T|\}$ ise bu vektörün etiketi $\mathcal{X}$ olarak değiştirilir, 3. adıma dönülür.

(b) Eğer $\alpha(i) \geq 0 \, \forall i \in \{1, 2...|T|\}$ ve $\alpha \neq \mathcal{O}$ ise bu vektörden ateşlenebilecek tüm geçişler ateşlenerek yeni M' işaretleme vektörleri elde edilir, $\mathcal{N}$ ile etiketlenir, 3. adıma dönülür.

(c) Eğer $\alpha(i) = 0 \; \forall i \in \{1, 2...|T|\}$ ise, $M = m_0$'dır ve m_0'dan M oluşana kadar olan yoldaki tüm işaretleme vektörleri PR kümesi içine atılır. K_x bir vektör olmak üzere;

$$K_x(p) = \max_{M' \in PR}(M'(p)), \forall p \in P \tag{4.8}$$

olarak belirlenir ve $K_{buf}=K_{buf} \cup K_x$ yapılarak 3.adıma dönülür.

4. $j = j + 1$ yapılır. Eğer $j < s + 1$ ise 2.adıma dönülür, $j \geq s + 1$ ise algoritmadan çıkılır.

Yöntem 2 için geliştirilen Yöntem2.m adlı Matlab programı Ek-D'de verilmiştir.

Teorem 4.3: Yöntem 2 Algoritmasında Eşitlik 4.7 ile elde edilen K sınır vektörü kısmi tersine dönüşebilir bir Petri ağının tersine dönüşebilirliğini garanti eden bir sınır vektörüdür.

İspat: Bir Petri ağında $\mathcal{O}$'dan farklı bir T-değişmezine karşılık gelen bir g geçiş dizisinin m_0'dan ateşlenebilmesi durumunda yeniden m_0'a ulaşılmaktadır. Bu esnada oluşan tüm işaretleme vektörleri m_0'a dönebilme özelliğine sahip olduğundan, $M \neq m_0$ şeklinde en az bir işaretleme vektörü oluşmuş ise Petri ağı kısmi tersine dönüşebilirdir. Herbir K_x vektörü, bu özellikteki bir geçiş dizisinin m_0'dan ateşlenmesi esnasında oluşan tüm işaretleme vektörlerini kapsayacak şekilde oluşturulmaktadır (bkz. Eşitlik 4.8). Eşitlik 4.7'de elde edilen K için $K(i) \geq K_x(i), \forall i \in \{1, 2..|P|\}$, $\forall K_x \in K_{buf}$ olduğundan K, Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme

vektörlerini kapsar ve ağın tersine dönüşebilirliğini garanti eder.

Örneğin. Şekil 4.1'deki Petri ağı için Yöntem2.m programı çalıştırıldığında. Algoritma-2 yardımıyla bulunan T-değişmezi $[1\ 1\ 0]^T$'dir. Yani "t_1, t_2" geçişlerinin birer kez kullanıldığı bir g geçiş dizisinin ($g = t_1t_2$ veya $g = t_2t_1$) m_0'dan ateşlenebilmesi durumunda m_0'a geri dönülür. Yöntem2.m programı ile böyle bir g geçiş dizisinin m_0'dan ateşlenebilirliği (ağın kısmi tersine dönüşebilirliği) araştırılmakta, eğer ateşlenebiliyorsa oluşan işaretleme vektörleri için K sınır vektörü belirlenmektedir. Bu Petri ağı için Yöntem2.m programı şu şekilde çalışmaktadır:

- $\rho(m_0, t_1)$=$[1\ 2\ 1]^T = M_1$, $GY(M_1) = [1\ 0\ 0]^T$, $\alpha = [0\ 1\ 0]^T$: M_1'den ateşlemelere devam edilir.
- $\rho(m_0, t_3)$=$[3\ 0\ 0]^T = M_2$, $GY(M_2) = [0\ 0\ 1]^T$, $\alpha = [1\ 1\ -1]^T$: M_2'den ateşlemelere devam edilmez.
- $\rho(M_1, t_1)$=$[0\ 3\ 2]^T = M_3$, $GY(M_3) = [2\ 0\ 0]^T$, $\alpha = [-1\ 1\ 0]^T$: M_3'den ateşlemelere devam edilmez.
- $\rho(M_1, t_2)$=$[2\ 1\ 0]^T = m_0$, $GY(m_0) = [1\ 1\ 0]^T$, $\alpha = [0\ 0\ 0]^T$: bir g geçiş dizisi ateşlenmiştir.
- $\rho(M_1, t_3)$=$[2\ 1\ 1]^T = M_4$, $GY(M_4) = [1\ 0\ 1]^T$, $\alpha = [0\ 0\ -1]^T$: M_4'den ateşlemelere devam edilmez.

Yukarıda da görüldüğü gibi bu Petri ağı için Mark&Silva Algoritması ile bulunan ve m_0'dan ateşlenmesiyle yeniden m_0'a ulaşan tek bir g geçiş dizisi ve dolayısıyla, bu geçiş dizisinin oluşturduğu tek bir yol vardır. Yöntem2.m programının çıktı dosyası, kullanılan T-değişmezlerini, bu T-değişmezlerinin ifade ettiği geçiş dizilerinden m_0'dan ateşlenmesi ile yeniden m_0'a ulaşanların oluşturduğu yoldaki işaretleme vektörlerini, her bir yol için bulunan K_x sınır vektörlerini ve Petri ağının tersine dönüşebilirliğini garanti eden K sınır vektörünü içermektedir (bkz. Şekil 4.3).

T-degismezi	Yol:	K_x
1 1 0	1 2 1	2 2 1
	2 1 0	
K		
2 2 1		

Şekil 4.3: Şekil 4.1 için Yöntem2.m programının çıktı dosyası

4.3 K Sınır Vektörü ile Konrolör Tasarımı

Sınırsız bir Petri ağı için bu bölümde sunulan yöntemlerden biri ile (Yöntem 1 veya Yöntem 2) bir K sınır vektörü elde edilebiliyorsa ağ kısmi tersine dönüşebilirdir ve K, Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesindeki tüm işaretleme vektörlerini kapsar.

Sınırsız bir Petri ağı için elde edilen K sınır vektörü [11]'de geliştirilen Bounded-Set algoritmasına girdi olarak verildiğinde, ağın ulaşılabilirlik kümesinin K sınırlı, kısmi tersine dönüşebilir bir alt kümesi (R_B) elde edilmektedir. R_B kümesi Reversible-Set algoritmasına girdi olarak verildiğinde elde edilen R_r kümesi tersine dönüşebilirdir. Eşitlik 4.9'da bu veriler kullanılarak tasarlanan kontrolör, $c : R(G, m_0) \times T \rightarrow \{0.1\}$ ile Petri ağının tersine dönüşebilirliği garanti edilmektedir [11].

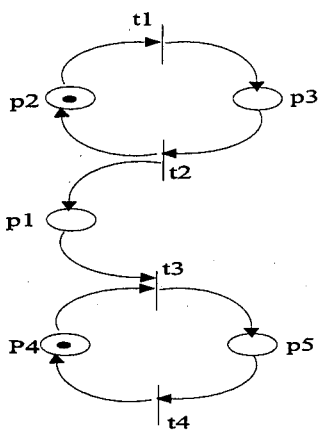
$$c(M,t) = \begin{cases} 1, & \text{eğer } \rho(M.t) \in R_r \\ 0, & \text{diğer durumlar} \end{cases} \tag{4.9}$$

Burada $c(M,t) = 1$, M işaretleme vektöründen t geçişin ateşlenebilmesi ve $c(M,t) = 0$, M işaretleme vektöründen t geçişinin ateşlenememesi anlamını taşır.

5. UYGULAMALAR

Bölüm 4'de kısmi tersine dönüşebilir bir Petri ağının tersine dönüşebilirliğini garanti eden bir K sınır vektörünün bulunabilmesi için kullanılabilecek iki yöntem ortaya kondu. Bu bölümde çeşitli Petri ağları için bu yöntemlerin uygulaması yapılacaktır.

5.1 Uygulama 1



Şekil 5.1: 1.Uygulama için Petri ağı

Şekil 5.1'deki Petri ağı [12] için giriş, çıkış matrisleri ve başlangıç işaretleme vektörü sırasıyla şu şekildedir;

$$N = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad O = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$m_0 = [0\ 0\ 1\ 1\ 0]$$

5.1.1 1.Uygulama İçin Yöntem 1

$rank(A) \neq |T|$ olduğundan bu ağın $\mathcal{O}$'dan farklı T-değişmezi vardır. ağ en az bir başlangıç vektörü için kısmi tersine dönüşebilirdir. Bu uygulama için Yöntem1.m programının çıktı dosyası Şekil 5.2'de görülmektedir.

```
R                    R'      K
00110  11010         00101   11111
10110  01001         01010
00101  01010
11001  21010
10101  20110
31010  21001
```

Şekil 5.2: 1.Uygulama için Yöntem1.m programının çıktı dosyası

5.1.2 1.Uygulama İçin Yöntem 2

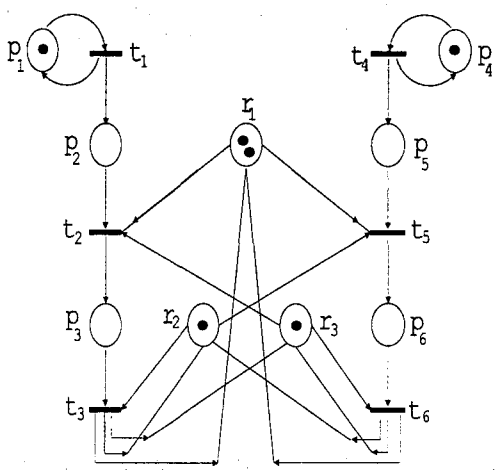
Bu Petri ağı için Tinv.m programı çalıştırıldığında bulunan T-değişmezi $[1\ 1\ 1\ 1]^T$ şeklindedir. Yani "t_1, t_2, t_3, t_4" geçişlerinin birer kez kullanıldığı bir g geçiş dizisinin m_0'dan ateşlenebilmesi durumunda m_0'a geri dönülür. Bu Petri ağı için bu özellikte iki ayrı geçiş dizisi m_0'dan ateşlenebilmektedir. Dolayısıyla bu ateşleme esnasında iki ayrı yol oluşmaktadır. Bu yollarda oluşan işaretleme vektörlerinin, her yol için elde edilen K_x sınır vektörlerinin ve Petri ağının tersine dönüşebilirliğini garanti eden K sınır vektörünün bulunduğu. Yöntem2.m programının çıktı dosyası Şekil 5.3'de görülmektedir.

T-degismezi	Yol:	K_x
1 1 1 1	0 0 1 0 1	1 1 1 1 1
	1 0 1 1 0	
	1 1 0 1 0	
	0 0 1 1 0	
	Yol:	K_x
	0 1 0 1 0	1 1 1 1 1
	0 1 0 0 1	
	1 1 0 1 0	
	0 0 1 1 0	

K
1 1 1 1

Şekil 5.3: 1.Uygulama için Yöntem2.m programının çıktı dosyası

5.2 Uygulama 2



Şekil 5.4: 2.Uygulama için Petri ağı

Şekil 5.4'deki Petri ağı [11] için ($P = \{p_1, p_2, p_3, r_1, r_2, r_3, p_4, p_5, p_6\}$) giriş, çıkış matrisleri ve başlangıç işaretleme vektörü sırasıyla şu şekildedir;

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad O = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$m_0 = [1\ 0\ 0\ 2\ 1\ 1\ 1\ 0\ 0]^T$$

5.2.1 2.Uygulama İçin Yöntem 1

$rank(A) \neq |T|$ olduğundan bu ağın $\mathcal{O}$'dan farklı T-değişmezi vardır. ağ en az bir başlangıç vektörü için kısmi tersine dönüşebilirdir. Bu uygulama için Yöntem1.m programının çıktı dosyası Şekil 5.5'de görülmektedir.

5.2.2 2.Uygulama İçin Yöntem 2

Bu Petri ağı için Tinv.m programı çalıştırıldığında $[1\ 1\ 1\ 0\ 0\ 0]^T$ ve $[0\ 0\ 0\ 1\ 1\ 1]^T$ olmak üzere iki tane T-değişmezi bulunur. Yani "t_1, t_2, t_3" ve/veya "t_4, t_5, t_6" geçişlerinin birer kez kullanıldığı geçiş dizilerinin, yani sırasıyla g_1 ve/veya g_2'nin m_0'dan ateşlenebilmesi durumunda m_0'a geri dönülür. Bu Petri ağı için g_1 ve g_2 özelliklerinde birer geçiş dizisi ateşlenebilmektedir. Dolayısıyla bu ateşleme esnasında iki yol oluşmaktadır. Bu yollarda oluşan işaretleme vektörlerinin, her yol için elde edilen K_x sınır vektörlerinin ve Petri ağının tersine dönüşebilirliğini garanti eden K sınır vektörünün bulunduğu, Yöntem2.m programının çıktı dosyası Şekil 5.6'da görülmektedir.

```
R                                   R'          K
100211100 110211100 101110100 111211111
100211110 120211100 100101101
101110100 110211110
100211120 100101101
111110100 101110110
120211110 110211120
110101101 100101111
111110110 101110120
101000101 120101101
110101111 111000101
101000111 130211100
100211130 121110100
130211110 120211120
110211130 100101121
121110110 111110120
101110130 130101101
120101111 110101121
121000101 111000111
101000121 140211100
100211140 131110100
140211110 130211120
120211130 110211140
100101131 131110110
121110120 111110130
101110140 140101101
130101111 120101121
110101131 131000101
121000111 111000121
101000131
```

Şekil 5.5: 2.Uygulama için Yöntem1.m programının çıktı dosyası

```
T-degismezi     Yol:            Kx
111000          101110100       111211100
                110211100
                100211100

T-degismezi     Yol:            Kx
000111          100101101       100211111
                100211110
                100211100

K
111211111
```

Şekil 5.6: 2.Uygulama için Yöntem2.m programının çıktı dosyası

6. SONUÇLAR

Bu çalışmada, sınırsız bir Petri ağının tersine dönüşebilirliğini garanti eden bir K sınır vektörünün bulunması için Yöntem 1 ve Yöntem 2 olarak adlandırılan iki yöntem geliştirilmiş, bu yöntemlerin Matlab'da oluşturulan Yöntem1.m ve Yöntem2.m programlarıyla gerçeklemeleri yapılmıştır.

Yöntem 1. m_0'a tek bir geçiş ateşlemesiyle ulaşan işaretleme vektörlerinin bulunduğu, Petri ağının R' kümesini ve m_0'dan bu elemanlara ulaşmak için ateşlenen geçiş dizilerindeki tüm işaretleme vektörlerini kapsayan sınır vektörünü Petri ağının tersine dönüşebilirliğini garanti eden sınır vektörü olarak belirlemektedir. Yöntem 2 ise, Mark&Silva Algoritması ile bulunan T-değişmezlerine ait geçiş dizilerinden m_0'dan ateşlenebilenleri bulmaktadır. Bu ateşlemeler esnasında elde edilen tüm işaretleme vektörlerini kapsayan bir K sınır vektörünü Petri ağının tersine dönüşebilirliğini garanti eden sınır vektörü olarak belirlemektedir.

Yöntem 1'de Petri ağının kısmi tersine dönüşebilir olup olmadığı kesin olarak belirlenir. Yöntem 1, bir sınır vektörü öneriyorsa ağ kısmi tersine dönüşebilirdir, önermiyorsa ağ kısmi tersine dönüşebilir değildir. Yöntem 2 ise tek yönlüdür. Öyle ki, bir sınır vektörü öneriliyorsa ağ kısmi tersine dönüşebilirdir, önerilmiyorsa kesin bir yargıya varılamaz. Çünkü Yöntem 2'de sadece Mark&Silva Algoritması ile bulunan T-değişmezlerine karşılık gelen geçiş dizilerinin m_0'dan ateşlenebilirliği kontrol edilmektedir. Oysa ki, Petri ağının $\mathcal{O}$'dan farklı bir T-değişmezi varsa, sonsuz tane T-değişmezi vardır.

Bu tezde geliştirilen her iki yöntemle bulunan sınır vektörleri Petri ağının tersine dönüşebilirliğini garanti eden sınır vektörleridir. Öyle ki, bulunan sınır vektörleri, [11]'de geliştirilen Bounded-Set algoritmasına girdi olarak verildiğinde ilgili Petri ağının ulaşılabilirlik kümesinin sınırlı bir alt kümesi (R_B) elde edilir. R_B Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesini, R_r, kapsar. Bu kümenin elde edilmesi için yine [11]'de geliştirilen Reversible-Set algoritması kullanılmaktadır. Bölüm 4.3'de, bu veriler temel

alınarak tasarlanan kontrolör Petri ağının tersine dönüşebilirliğini garanti etmektedir.

Üretim sistemlerinde gereksiz stoklama, gereksiz kaynak kullanımı ve başlangıç durumuna dönememe; yer, zaman ve işgücü kaybına sebep olarak sistemin kontrolünü zorlaştırır, verimini azaltır. Bunun için, Petri ağı ile modellenmiş bir üretim sisteminin sınırlılık ve tersine dönüşebilir özellik taşıması, bu sistemlerin yüksek performans gösterebilmesi için aranan şartlardandır. Petri ağı ile modellenmiş bir sistemde, bu çalışmada belirlenen sınır vektörleri kullanılarak, Petri ağının ulaşılabilirlik kümesinin tersine dönüşebilir bir alt kümesi elde edilir.

Bundan sonraki çalışmalarda Yöntem 1 kullanılarak sınırsız ağların ulaşılabilirlik kümesinin kısmi tersine dönüşebilir bir alt kümesi bulunduktan sonra, daha önce yapılan çalışmalardan faydalanılarak bu küme üzerinde tutarlılık, canlılık analizleri yapılabilir.

KAYNAKLAR

[1] AYBAR, A.. *Petri Ağlarında Örtüşmeli Ayrıştırma ve Genleştirme Kullanılarak Kontrolör Tasarımı*. Doktora Tezi, Anadolu Üniversitesi, Fen Bilimleri Enstitüsü, Eskişehir, Türkiye (2001).

[2] HO, Y., *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World*, The Institute of Electrical and Electronics Engineering, Inc., Newyork (1992).

[3] ZHOU, M. ve DICESARE, F.. *Petri Net Synthesis For Discrete Event Control of Manufacturing Systems*, Norwell, Massachusetts: Kluwer Academic Publishers (1993).

[4] PROTH, J. ve XIE, X., *Petri Nets*, John Wiley & Sons, West Sussex (1996).

[5] KARP, R. M. ve MILLER, R. E., *Parallel program schemata*, Journal of Computer System Science, **3**, 147–195 (1969).

[6] SIFAKIS, J., *Structural properties of Petri nets*, Mathematical Foundation of Computer Science, **65**, 474–483 (1978).

[7] MARTINEZ. J. ve SILVA, M.. *A simple and fast algorithm to obtain all invariants of a generalized Petri net*, In Applications and Theory of Petri Nets, **52**, 301–311 (1982).

[8] CHEN, Y., TSAI, W. T ve CHAO, D., *Dependency analysis: A Petri net based technique for synthesis large concurrent systems*, IEEE transactions on Parallel and Distributed Systems, **4**, 414–426 (1993)

[9] YE, X., ZHOU, J. ve SONG, X., *On reachability graphs of Petri nets*, Computers & Electrical Engineering, **29**, 263–272 (2003).

[10] AYBAR, A.. İFTAR, A. ve APAYDIN, H., *Centralized and decentralized supervisory controller design to enforce boundednesss, liveness and reversibility in Petri nets*. (Yayınlanmak üzere gönderildi).

[11] AYBAR. A. ve İFTAR, A., *Controller design to enforce boundedness. liveness. and reversibility in Petri nets*, Preprints of the 7th IFAC Workshop on Intelligent Manufacturing Systems, 199–204 (2003).

[12] DESROCHERS. A. A. ve AL-JAAR, R. Y.. *Applications of Petri Nets in Manufacturing Systems*. The Institute of Electrical and Electronics Engineers. Inc., New York, (1995).

[13] PETERSON. J. L. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs. NJ : Prentice-Hall. New Jersey (1981).

[14] RAFIEI. D., *Enterprise Modeling in Manufacturing*. Yüksek Lisans Tezi. University of Waterloo, Canada (1994).

[15] ANTON. H., *Elementary Linear Algebra*, John Wiley Sons, Inc.. Newyork (1994).

[16] CASSANDRAS ve LAFORTUNE, *Introduction to Discrete Event Systems*, Kluwer, Norwell, MA (1999).

EK-A

"cover.m" programı çalıştırılmadan önce, Petri ağının Matlab komut penceresinde çağrılan bir girdi dosyası (bkz. Ek-E) ile tanımlanması gerekir. Program bu işlemin ardından çağrılır.

ALGORITMA-1 (Kapsayabilirlik ağacı algoritması) için cover.m programı

```
TK=[];
R=[m0'];
Txmat=[];
Resmat=[];
Resmat1=[];
Bmat=[];
Oldmat=[];
[xnum,ynum]= size(N);
t=0;
Mmat=[];
check1=0;
while check1~=1
  [a,b]=size(R);
  Rmat=[];
   if a>0
     for j=1:b,
       R(:,j)';
       [Tr]= Ea(N,O,R(:,j)');
  [x,y]=size(Tr);
       [qnum,wnum]=size(Mmat);
       same=0;
     for i=1:y,
      Tr(i);
```

```
    M=rho(N,O,R(:,j)',Tr(:,i));
        Txmat=[Txmat
    Tr(i)];
Bmat=[Bmat
      R(:,j)'];
   Resmat=[Resmat
      M];
   Oldmat=[Oldmat
           O];
   [xx,yy]=size(Bmat);
   [qq,ww]=size(Resmat);
   Cmat=M;
   Ymat=[];
   tx=0;
   while tx<1,
    for t=1:qq,
  Cmat;
  Resmat(t,:);
   if Cmat==Resmat(t,:)
       Cmat=Bmat(t,:);
       Ymat=[Ymat
       Bmat(t,:)];
    t=qq;
    break;
  end
 end
 if Cmat==mO
  tx=1;
 break;
 end
```

```
 end
Ymat;
[aa,bb]=size(Ymat);
     for l=1:aa,
     same=0;
     for v=1:xnum,
        if M(1,v)>=Ymat(l,v)
           same=same+1;
        end
     end
     if same==xnum
        for v=1:xnum,
           if M(1,v)>Ymat(l,v)
              M(1,v)=100;
           end
        end
     end
  end

  M;
  for l=1:qq,
    same=0;
    same1=0;
    for v=1:xnum,
        if M(1,v)==Bmat(l,v) %| M(1,v)==Resmat(l,v)
           same=same+1;
        end
     end
     if same==xnum
  Oldmat(qq)=1;
```

```
            for v=1:xnum,
            if M(1,v)==100
               M(1,v)=200;
          end
        end
      end
  end

[yy,ll]=size(Rmat);
Rmat1=Rmat';
if ll>0
  for l=1:ll,
    same=0;
      for v=1:xnum,
        if M(1,v)==Rmat1(l,v)
            same=same+1;
        end
      end
      if same==xnum
  Oldmat(qq)=1;
            for v=1:xnum,
            if M(1,v)==100
               M(1,v)=200;
          end
        end
      end
  end
  end

if Oldmat(qq)==0
```

```
        Rmat=[Rmat M'];
    end
    Resmat1=[Resmat1
          M];
    Resmat=Resmat1;
 end
end
R=Rmat;
t=t+1;
else
   check1=1;
end
end
tree=[m0];
[xx,yy]=size(Resmat);
for i=1:xx
   if Oldmat(i,1)==0
         tree=[tree
         Resmat(i,:)];
   end
end

Dead=[];
[dx,dy]=size(Resmat);
for i=1:dx
   Resmat(i,:);
   [Trd]= Ea(N,0,Resmat(i,:));
   [xa,ya]=size(Trd);
   [dxx,dyy]=size(Dead);
      if ya==0
```

```
        if dxx==0
           Dead=[Dead ;Resmat(i,:)];
        else
           samedd=0;
           for ii=1:dxx
              if Dead(ii,:)==Resmat(i,:);
                 samedd=1;
              end
           end
         if samedd==0
           Dead=[Dead ;Resmat(i,:)];
           [dxx,dyy]=size(Dead);
         end
        end
     end
end

   Txmatt=Txmat';
   Tdead=[];
   [tx,ty]=size(Tdead);
   [x1,y1]=size(Txmatt);
   [x,y]=size(TK);
   for i=1:y
      samet=0;
      for j=1:y1
         if TK(1,i)==Txmatt(1,j)
            samet=1;
         end
      end
      if samet==0
```

```
        [tx,ty]=size(Tdead);
           if ty==0
              Tdead=[Tdead TK(1,i)];
           else
              samed=0;
              for ii=1:ty
                 if Tdead(1,ii)==TK(1,i)
                    samed=1;
                 end
              end
                 if samed==0
                    Tdead=[Tdead TK(1,i)];
                    [tx,ty]=size(Tdead);
                 end
              end
           end
        end

fid=fopen('result.m','w');
fprintf(fid,'%s','Bmat');
fprintf(fid,'\t\t');
fprintf(fid,'%s','Resmat');
fprintf(fid,'\t\t');
fprintf(fid,'%s','Txmat');
fprintf(fid,'\t');
fprintf(fid,'%s','Oldmat');
fprintf(fid,'\n');
fprintf(fid,'\n');
[a,b]=size(Bmat);
for i=1:a,
```

```
   for j=1:xnum,
      if Bmat(i,j)~=100 & Bmat(i,j)~=200
         fprintf(fid,'%d',Bmat(i,j));
      else
          fprintf(fid,'%s','w');
       end
   end
  fprintf(fid,'\t\t');
  for j=1:xnum,
      if Resmat(i,j)~=100 & Resmat(i,j)~=200
         fprintf(fid,'%d',Resmat(i,j));
      else
          fprintf(fid,'%s','w');
       end
   end
   fprintf(fid,'\t\t');
   fprintf(fid,'%d',Txmat(i,:));
   fprintf(fid,'\t\t');
   fprintf(fid,'%d',Oldmat(i,:));
   fprintf(fid,'\n');
   end
[dx,dy]=size(Dead);
fprintf(fid,'\n\n');
fprintf(fid,'%s','DEAD MARKINGS');
fprintf(fid,'\n');
if dx==0
   fprintf(fid,'%s','No Dead Markings');
else
   for i=1:dx,
   for j=1:dy,
```

```
      if Dead(i,j)~=100 & Dead(i,j)~=200
         fprintf(fid,'%d',Dead(i,j));
      else
          fprintf(fid,'%s','w');
       end
    end
    fprintf(fid,'\n');
 end
end

[dx,dy]=size(Tdead);
fprintf(fid,'\n\n');
fprintf(fid,'%s','DEAD TRANSITIONS');
fprintf(fid,'\n');
if dy==0
   fprintf(fid,'%s','No Dead Transitions ');
else
for i=1:dy
   fprintf(fid,'%d',Tdead(1,i));
   fprintf(fid,'\t\t');
end
end
fclose(fid);
[a,b]=size(Resmat);
Wmat=[];
for i=1:a,
   for j=1:b,
      if Resmat(i,j)==100 | Resmat(i,j)==200
         [c,d]=size(Wmat);
         same=0;
```

```
        for k=1:c,
            if Wmat(k,:)== Resmat(i,:)
                same=same+1;
            end
        end
        if same==0
            Wmat=[Wmat
           Resmat(i,:)];
         break;
       end
     end
   end
end
[q,w]=size(Wmat);
Tsize=[];
for i=1:q,
   wnum=0;
   for j=1:w,
      if Wmat(i,j)==100 | Wmat(i,j)==200
         wnum=wnum+1;
      end
   end
   Tsize=[Tsize
      wnum];
end
maxnum=max(Tsize);
Wmat2=[];
Wmat1=[];
for i=1:q,
   if Tsize(i)==maxnum
```

```
      Wmat2=[Wmat2
         Wmat(i,:)];
   else
      Wmat1=[Wmat1
         Wmat(i,:)];
   end
end
Tn=[];
Tw=[];
for j=1:w,
   if Wmat2(1,j)==100 | Wmat2(1,j)==200
      Tw=[Tw
         j];
   else
      Tn=[Tn
         j];
   end
end
[a,b]=size(Wmat2);
[c,d]=size(Tw);
Wend=[];
for i=1:a,
   for j=1:c,
      Wmat2(i,Tw(j))=300;
   end
   [q,w]=size(Wend);
   same=0;
   for k=1:q,
      if Wmat2(i,:)==Wend(k,:)
         same=1;
```

```
        end
    end
    if same==0
        Wend=[Wend
            Wmat2(i,:)];
    end
end
[xx,yy]=size(Wend);
snr=Tw;
```

EK-B

"Tinv.m" programı çalıştırılmadan önce, Petri ağının Matlab komut penceresinde çağrılan bir girdi dosyası (bkz. Ek-E) ile tanımlanması gerekir. Program, bu işlemin ardından çağrılır.

ALGORITMA-2 (Mark&Silva algoritması) için Tinv.m programı

```
A=O-N;
hanA=null(A);
if isempty(hanA)==0
[a,b]=size(A');
AA=[eye(a) A'];
[c,d]=size(AA);
for han=a+1:d,
   han;
   T=[];
   BB=[];
  for j=1:c,
      for z=1:c,
         if AA(j,han)+AA(z,han)==0 & AA(j,han)~=0 & AA(z,han)~=0
            AA(j,:);%
            AA(z,:);%
            aa=AA(j,:)+AA(z,:);%
            BB=[BB;aa];
            T=[T j z];
         end
      end
   end

   Tx=[];
```

```
same=0;
[e f]=size(Tx);
[g h]=size(T);
for j=1:h,
   for z=1:f,
      if Tx(1,z)==T(1,j)
         same=1;
         break
      end
   end
    if same ~=1
       Tx=[Tx T(1,j)];
       [e f]=size(Tx);
    end
    same=0;
end

[e f]=size(Tx);
AA=[AA;BB];
[fa,fb]=size(AA);
CC=[];
flag=0;
for ii=1:fa,
   flag=0;
   for jj=1:f,
      if ii==Tx(1,jj)
         flag=1;
      end
   end
   if flag==0
```

```
        CC=[CC;AA(ii,:)];
    end
 end
 AA=CC;

    same=0;
    [c,d]=size(AA);
    Ax=[];
    [e,f]=size(Ax);
    for i=1:c,
      for j=1:e,
          if AA(i,:)==Ax(j,:)
             same=1;
             break
          end
       end
       if same==0
          Ax=[Ax ;AA(i,:)];
       end
      [e,f]=size(Ax);
      same=0;
   end
   AA=Ax;
   [c,d]=size(AA);
end

[a,b]=size(A');
Tin=[];
[c,d]=size(AA);
for i=1:c,
```

```
      b=AA(i,a+1:d);
         if b==0
         Tin=[Tin;AA(i,1:a)];
      end
   end
else
end
```

EK-C

"Yöntem1.m" programı çalıştırılmadan önce, Petri ağının Matlab komut penceresinde çağrılan bir girdi dosyası (bkz. Ek-E) ile tanımlanması gerekir. Program, bu işlemin ardından çağrılır.

Yöntem1.m programı

```
A=O-N;
[P,T]=size(N);
if rank(A)==T
   disp('PA ktd. degildir');
else
   cover;       % ALGORITMA-1
   FindRO;      % ALGORITMA-3
   FindRtilde;   % ALGORITMA-4

  Rp=[];   %R' kmesinin olusturulmasi
  [aa,bb]=size(Rtilde);
  for i=1:aa
     [Tr]= Ea(N,O,Rtilde(i,:));
     [aaa,bbb]=size(Tr);
     for ii=1:bbb
        M=rho(N,O,Rtilde(i,:),Tr(:,ii));
        if M==mO
           Rp=[Rp;Rtilde(i,:)];
        end
     end
  end
  [aq,bq]=size(Rp);
  a=0;
  for ii=1:aq,
```

```
      if Rp(ii,:)~=m0
         a=1;
      end
   end

if a==0
    disp('PA ktd. degildir');
else
  FindMin_K;  %ALGORITMA-5
end
end

fid=fopen('OutputFile1.m','w');
fprintf(fid,'%s','Rtilde:    ');
fprintf(fid,'\n\n');
[a,b]=size(Rtilde);
for i=1:a,
   for j=1:b,
      fprintf(fid,'%d',Rtilde(i,j));
   end
   fprintf(fid,'\n');
end
fprintf(fid,'\n\n');
fprintf(fid,'%s','Rp:    ');
fprintf(fid,'\n\n');
[a,b]=size(Rp);
for i=1:a,
   for j=1:b,
      fprintf(fid,'%d',Rp(i,j));
   end
```

```
   fprintf(fid,'\n');
end
fprintf(fid,'\n\n');
fprintf(fid,'%s','K:     ');
fprintf(fid,'\n\n');
[a,b]=size(K);

for i=1:a,
   for j=1:b,
      fprintf(fid,'%d',K(i,j));
   end
   fprintf(fid,'\n');
end
fclose(fid);
```

ALGORITMA-3 (R_0 oluşturma algoritması) için FindR0.m programı

```
R=[m0'];
Txmat=[];
Resmat=[];
Bmat=[];
Oldmat=[];
Stepmat=[];
Inmat=[];
Txx=[];
Rx=[];
[xnum,ynum]= size(N);
Inmat=[1];
t=0;
Mmat=[];
```

```
check1=0;
Step=0;
XX=0;
ind=1;
while check1~=1
   [a,b]=size(R);
   Step=Step+1;
   Rmat=[];
   Rxmat=[];
   if a>0
   for j=1:b,
   R(:,j)';
   Trx=[];
   [Tr]= Ea(N,O,R(:,j)');
[x,y]=size(Tr);
   Trx=[Tr zeros(1,xnum-y)];
   [qnum,wnum]=size(Mmat);
   same=0;
   for i=1:y,
      Tr(i);
      M=rho(N,O,R(:,j)',Tr(:,i));
      ind=ind+1;
      Txmat=[Txmat
          Tr(i)];
         Bmat=[Bmat
            R(:,j)'];
         Resmat=[Resmat
            M];
         Oldmat=[Oldmat
                O];
```

```
    Stepmat=[Stepmat
       Step];
    Inmat=[Inmat
       ind];
    Txx=[Txx
       Trx];
[xx,yy]=size(Bmat);
[qq,ww]=size(Resmat);
[aa,bb]=size(M);
for tw=1:qq-1,
   same=0;
   if M==Resmat(tw,:)
      Oldmat(qq)=2;
      break;
   end
end
 for k=1:xx,
  sims=0;
  for w=1:bb,
      if Bmat(k,w)==M(:,w)
         sims=sims+1;
      end
   end
   if sims==bb
      Oldmat(qq)=2;
      break;
   end
   sim=0;
   for w=1:bb,
      if M(:,w)>=Bmat(k,w)
```

```
                sim=sim+1;
            end
        end
        if sim==bb
            sim1=0;
            [Trv]= Ea(N,0,M);
[xv,yv]=size(Trv);
    Trxv=[Trv zeros(1,xnum-yv)];
                for v=1:xnum,
                if Txx(k,v)==Trxv(:,v)
                  sim1=sim1+1;
                end
            end
            if sim1==xnum
            Oldmat(qq)=1;
            break;
        end
        end
        end
      if Oldmat(qq)==0
        Rmat=[Rmat
        M];
      end
      [ax,bx]=size(Rx);
      [ay,by]=size(Rxmat) ;
      if Oldmat(qq)==1
        for n=1:ax,
            if M==Rx(n,:)
              Oldmat(qq)=2;
            end
```

```
            end
           for m=1:ay,
               if M==Rxmat(m,:)
                  Oldmat(qq)=2;
               end
            end
          end
         if Oldmat(qq)==1
            Rxmat=[Rxmat
               M];
        end
        end
      end
  R=Rmat';
  Rx=[Rx
     Rxmat];
end
[I,J]=find(Oldmat==0);
if Stepmat(max(I))<Stepmat(qq)
   check1=1;
end
if isempty(I)==1
   check1=1;
end
end
RO=[];
RO=[mO];
[ax,bx]=size(Resmat);
[cx,dx]=size(RO);
for i=1:ax,
```

```
same=0;
for j=1:cx,
   if Resmat(i,:)==RO(j,:)
      same=1;
   end
end
if same==0
      RO=[RO
         Resmat(i,:)];
       [cx,dx]=size(RO);
   end
end
```

ALGORITMA-4 ($\tilde{R}$ oluşturma algoritması)için FindRtilde.m programı

```
R=[Rx'];
Txmat1=[];
Resmat1=[];
Bmat1=[];
Oldmat1=[];
Stepmat1=[];
Txx1=[];
Rx1=[];
[xnum,ynum]= size(N);
Inmat1=[ind];
t=0;
Mmat=[];
check1=0;
 Step=0;
 while check1~=1
```

```
   [a,b]=size(R);
   Step=Step+1;
   Rmat=[];
   Rxmat1=[];
   if a>0
   for j=1:b,
   R(:,j)';
   Trx=[];
   [Tr]= Ea(N,O,R(:,j)');
[x,y]=size(Tr);
   Trx=[Tr zeros(1,xnum-y)];
   [qnum,wnum]=size(Mmat);
   same=0;
   for i=1:y,
      Tr(i);
      M=rho(N,O,R(:,j)',Tr(:,i));
      ind=ind+1;
      Txmat1=[Txmat1
          Tr(i)];
         Bmat1=[Bmat1
            R(:,j)'];
         Resmat1=[Resmat1
            M];
         Oldmat1=[Oldmat1
                O];
             Stepmat1=[Stepmat1
                Step];
             Inmat1=[Inmat1
                ind];
             Txx1=[Txx1
```

```
        Trx];
[xx,yy]=size(Bmat1);
[qq,ww]=size(Resmat1);
[aa,bb]=size(M);
[aq,bq]=size(RO);
for tw=1:qq-1,
    if M==Resmat1(tw,:)
      Oldmat1(qq)=2;
      break;
   end
  end
  for ty=1:aq,
     if M==RO(ty,:)
      Oldmat1(qq)=2;
      break;
   end
  end
for k=1:xx,
  sims=0;
  for w=1:bb,
      if Bmat1(k,w)==M(:,w)
         sims=sims+1;
      end
   end
   if sims==bb
      Oldmat1(qq)=2;
      break;
   end
  sim=0;
   for w=1:bb,
```

```
            if M(:,w)>=Bmat1(k,w)
               sim=sim+1;
            end
         end
         if sim==bb
            sim1=0;
            [Trv]= Ea(N,O,M);
[xv,yv]=size(Trv);
    Trxv=[Trv zeros(1,xnum-yv)];
          for v=1:xnum,
              if Txx1(k,v)==Trxv(:,v)
                 sim1=sim1+1;
              end
            end
            if sim1==xnum
            Oldmat1(qq)=1;
            break;
         end
         end
        end
       if Oldmat1(qq)==0
         Rmat=[Rmat
         M];
      end
      [ax,bx]=size(Rx1);
      [ay,by]=size(Rxmat1);
      if Oldmat1(qq)==1
         for n=1:ax,
            if M==Rx1(n,:)
               Oldmat1(qq)=2;
```

```
            end
        end
        for m=1:ay,
            if M==Rxmat1(m,:)
                Oldmat1(qq)=2;
            end
        end
        end
      if Oldmat1(qq)==1
        Rxmat1=[Rxmat1
            M];
      end
     end
   end
  R=Rmat';
  Rx1=[Rx1
     Rxmat1];
end
[I,J]=find(Oldmat1==0);
if Stepmat1(max(I))<Stepmat1(qq)
   check1=1;
end
if isempty(I)==1
   check1=1;
end
end
RO1=[];
RO1=[];
[ax,bx]=size(Resmat1);
[cx,dx]=size(RO1);
```

```
[wx,wy]=size(R0);
for i=1:ax,
   same=0;
   for j=1:cx,
      if Resmat1(i,:)==R01(j,:)
         same=1;
      end
   end
   same1=0;
    for j=1:wx,
      if Resmat1(i,:)==R0(j,:)
         same1=1;
      end
   end
    if same==0 & same1==0
         R01=[R01
            Resmat1(i,:)];
          [cx,dx]=size(R01);
      end
    end
  [x,y]=size(R01);
  [a,b]=size(snr);
  same=0;
 for i=1:x,
   for j=1:a,
      if m0(:,snr(j))-1 <= R01(i,snr(j)) & R01(i,snr(j))<=m0(:,snr(j))+1
         same=1;
      end
   end
end
```

```
  neigho=0;
if same==0
   neigho=0;
else
   neigho=1;
end
if neigho==1
   R=[Rx1'];
Txmat2=[];
Resmat2=[];
Bmat2=[];
Oldmat2=[];
Stepmat2=[];
Txx2=[];
Rx2=[];
[xnum,ynum]= size(N);
Inmat2=[ind];
t=0;
Mmat=[];
check1=0;
 Step=0;
 while check1~=1
   [a,b]=size(R);
   Step=Step+1;
   Rmat=[];
   Rxmat2=[];
   if a>0
   for j=1:b,
   R(:,j)';
   Trx=[];
```

```
   [Tr]= Ea(N,0,R(:,j)');
[x,y]=size(Tr);
   Trx=[Tr zeros(1,xnum-y)];
   [qnum,wnum]=size(Mmat);
   same=0;
   for i=1:y,
      Tr(i);
      M=rho(N,0,R(:,j)',Tr(:,i));
      ind=ind+1;
      Txmat2=[Txmat2
        Tr(i)];
         Bmat2=[Bmat2
            R(:,j)'];
         Resmat2=[Resmat2
            M];
         Oldmat2=[Oldmat2
                0];
             Stepmat2=[Stepmat2
                Step];

             Inmat2=[Inmat2
                ind];
             Txx2=[Txx2
                Trx];
         [xx,yy]=size(Bmat2);
         [qq,ww]=size(Resmat2);
         [aa,bb]=size(M);
         [aq,bq]=size(R0);
         [aq1,bq1]=size(R01);
         for tw=1:qq-1,
```

```
    if M==Resmat2(tw,:)
      Oldmat2(qq)=2;
      break;
   end
  end
  for ty=1:aq,
     if M==RO(ty,:)
      Oldmat2(qq)=2;
      break;
   end
  end
  for tq=1:aq1,
     if M==RO1(tq,:)
      Oldmat2(qq)=2;
      break;
   end
  end
 for k=1:xx,
    sims=0;
  for w=1:bb,
      if Bmat2(k,w)==M(:,w)
         sims=sims+1;
      end
   end
   if sims==bb
      Oldmat2(qq)=2;
      break;
   end
  sim=0;
   for w=1:bb,
```

```
                if M(:,w)>=Bmat2(k,w)
                    sim=sim+1;
                end
            end
            if sim==bb
                sim1=0;
                [Trv]= Ea(N,O,M);
[xv,yv]=size(Trv);
    Trxv=[Trv zeros(1,xnum-yv)];
            for v=1:xnum,
                if Txx2(k,v)==Trxv(:,v)
                    sim1=sim1+1;
                end
            end
            if sim1==xnum
            Oldmat2(qq)=1;
            break;
            end
            end
            end
        if Oldmat2(qq)==0
            Rmat=[Rmat
            M];
        end
        [ax,bx]=size(Rx2);
        [ay,by]=size(Rxmat2);
        if Oldmat2(qq)==1
            for n=1:ax,
                if M==Rx2(n,:)
                    Oldmat2(qq)=2;
```

```
                end
            end
            for m=1:ay,
                if M==Rxmat2(m,:)
                    Oldmat2(qq)=2;
                end
            end
            end
        if Oldmat2(qq)==1
            Rxmat2=[Rxmat2
                M];
        end
        end
    end
  R=Rmat';
  Rx2=[Rx2
     Rxmat2];
end
[I,J]=find(Oldmat2==0);
if Stepmat2(max(I))<Stepmat2(qq)
   check1=1;
end
if isempty(I)==1
   check1=1;
end
end
RO2=[];
[ax,bx]=size(Resmat2);
[cx,dx]=size(RO1);
[wx,wy]=size(RO);
```

```
[qx,qy]=size(RO2);
for i=1:ax,
   same=0;
   for j=1:cx,
      if Resmat2(i,:)==RO1(j,:)
         same=1;
      end
   end
   same1=0;
    for j=1:wx,
      if Resmat2(i,:)==RO(j,:)
         same1=1;
      end
   end
   same2=0;
    for j=1:qx,
      if Resmat2(i,:)==RO2(j,:)
         same2=1;
      end
   end
   if same==0 & same1==0 & same2==0
         RO2=[RO2
            Resmat2(i,:)];
          [qx,qy]=size(RO2);
      end
    end
   Rx2;
   RO2;
[x,y]=size(RO2);
[a,b]=size(snr);
```

```
same=0;
for i=1:x,
   for j=1:a,
      if m0(:,snr(j))-1 <= RO2(i,snr(j)) & RO2(i,snr(j))<=m0(:,snr(j))+1
         same=1;
      end
   end
end
neigho1=0;
if same==0
   neigho1=0;
else
   neigho1=1;
end
if neigho1==0 & neigho==1
   Rtilde=[RO
      RO1];
   Bson=[Bmat
      Bmat1];
   Resson=[Resmat
      Resmat1];
   Oldson=[Oldmat
      Oldmat1];
   Stepson=[Stepmat
      Stepmat1];
   Inson=[Inmat
      Inmat1];
 end
if neigho1==1
      Rtilde=[RO
```

```
        RO1
    RO2];
Bson=[Bmat
    Bmat1
    Bmat2];
Resson=[Resmat
    Resmat1
Resmat2];
Oldson=[Oldmat
    Oldmat1
Oldmat2];
Stepson=[Stepmat
    Stepmat1
Stepmat2];
Inson=[Inmat
    Inmat1
Inmat2];
end
else
    Rtilde=[RO];
    Bson=[Bmat
    ];
    Resson=[Resmat
    ];
    Oldson=[Oldmat
    ];
    Stepson=[Stepmat
    ];
    Inson=[Inmat
    ];
```

```
end
```

ALGORITMA-5 (Minimum K algoritması) için FindMin-K.m programı

```
[a,b]=size(Rp);
  [xx,yy]=size(Bson);
  [qq,ww]=size(Resson);
  Kmat=[];
  for i=1:a,
      check=0;
      Cmat=Rp(i,:);
      Yson=Cmat;
      bas=0;
     while check<1,
         for t=1:qq,
            if Cmat==Resson(t,:)
              Cmat=Bson(t,:);
             Yson=[Yson
             Bson(t,:)];
           t=qq;
           bas=1;
           break;
         end
       end
     if Cmat==m0
          check=1;
          Yson;
          K=max(Yson);
          Kmat=[Kmat
             K];
```

```
        break;
    end
 end
end
[c,d]=size(Kmat);
if c==1
   K=Kmat;
else
   K=max(Kmat);
end
```

EK-D

"Yöntem2.m" programı çalıştırılmadan önce, Petri ağının Matlab komut penceresinde çağrılan bir girdi dosyası (bkz. Ek-E) ile tanımlanması gerekir. Program, bu işlemin ardından çağrılır.

Yöntem2.m programı

```
A=O-N;
[P,T]=size(N);
if rank(A)==T
   disp('PA ktd. degildir');
else
   Tinv;      %ALGORITMA-2
   FindPath; %ALGORITMA-6
   [a,b]=size(Kx);
   if a==1
      K=Kx;
   else
      K=max(Kx);
   end
   end
```

ALGORITMA-6 (Yol Bulma algoritması) için FindPath.m programı

```
fid=fopen('OutputFile2.m','w');
[xnum,ynum]= size(N);
Kx=[];
Kmat=[];
[aq,bq]=size(Tin);
for iq=1:aq,
 C=Tin(iq,:);
```

```
 R=[m0'];
Txmat=[];
Resmat=[];
Resmat1=[];
Bmat=[];
Oldmat=[];
t=0;
Mmat=[];
check1=0;
while check1~=1
   [a,b]=size(R);
   Rmat=[];
   if a>0
   for j=1:b,
   R(:,j)';
   [Tr]= Ea(N,O,R(:,j)');
 [x,y]=size(Tr);
   [qnum,wnum]=size(Mmat);
   same=0;
   for i=1:y,
      Tr(i);
      M=rho(N,O,R(:,j)',Tr(:,i));
         [xx,yy]=size(Bmat);
         [qq,ww]=size(Resmat);
prev=0;
ind=0;
Ymat2=[];
Tmat2=[];
for q1=1:qq,
       if M==Resmat(q1,:)
```

```
            Cmat=Bmat(q1,:);
           Ymat2=[Ymat2
            Bmat(q1,:)];
           Tmat2=[Tmat2
             Txmat(q1)];
                prev=1;
                ind=q1;
  break;
 end
         end
if prev==1
  tx=0;
      while tx<1,
         for t=1:qq,
            Cmat;
            Resmat(t,:);
         if Cmat==Resmat(t,:) & Oldmat(t)== 0
            Cmat=Bmat(t,:);
            Ymat2=[Ymat2
               Bmat(t,:)];
            Tmat2=[Tmat2
               Txmat(t)];
          break;
        end
         end
         if Cmat==m0
          tx=1;
         break;
         end
         end
```

```
Ymat2;
Tmat2;
end
      Txmat=[Txmat
          Tr(i)];
      Bmat=[Bmat
            R(:,j)'];
         Resmat=[Resmat
            M];
         Oldmat=[Oldmat
                0];
          [xx,yy]=size(Bmat);
          [qq,ww]=size(Resmat);
        Cmat=R(:,j)';
        Ymat=[];
        Tmat=[];
        Ymat=[Ymat
           R(:,j)'];
        Tmat=[Tmat
           Tr(i)];
       tx=0;
       while tx<1,
          for t=1:qq,
             Cmat;
             Resmat(t,:);
          if Cmat==Resmat(t,:) & Oldmat(t)== 0
             Cmat=Bmat(t,:);
             Ymat=[Ymat
                Bmat(t,:)];
             Tmat=[Tmat
```

```
          Txmat(t)];
      break;
   end
    end
    if Cmat==m0
     tx=1;
    break;
    end
   end
 Ymat;
 Tmat;
Ymat2;
Tmat2;
Ymat;
Tmat;
E1=max(Ymat);
E2=max(Ymat2);
if isempty(E2)==0
 if E2>E1
    Oldmat(ind)=1;
    else
      Oldmat(qq)=1;
   end
end
[aa,bb]=size(Tmat);
Cma=zeros(1,ynum);
for ww=1:ynum,
    same=0;
     for tq=1:aa,
       if ww==Tmat(tq)
```

```
                same=same+1;
              end
            end
           Cma(1,ww)=same;
       end
       Cma ;
       neg=0;
       Diff=C-Cma;
       sames=0;
       for xa=i:ynum,
        ' if Diff(1,xa)<0
             neg=1;
          end
          if Diff==0
             neg=2;
          end
          end
       if neg==1
      Oldmat(qq)=1;
           end
           if neg==2
              Oldmat(qq)=2;
           end
    [yy,ll]=size(Rmat);
    Rmat1=Rmat';
     if Oldmat(qq)==0
       Rmat=[Rmat M'];
    end
  end
end
```

```
R=Rmat;
else
   check1=1;
end
end
fprintf(fid,'%s','Bmat');
for i=1:xnum,
   fprintf(fid,'\t');
end
fprintf(fid,'%s','Resmat');
for i=1:xnum,
   fprintf(fid,'\t');
end
fprintf(fid,'%s','Txmat');
fprintf(fid,'\t');
fprintf(fid,'%s','Oldmat');
fprintf(fid,'\n');
fprintf(fid,'\n');
[a,b]=size(Bmat);
for i=1:a,
   for j=1:xnum,
      if Bmat(i,j)~=100
         fprintf(fid,'%d',Bmat(i,j));
      else
          fprintf(fid,'%s','w');
       end

   end
  fprintf(fid,'\t\t');
  for j=1:xnum,
```

```
      if Resmat(i,j)~=100 & Resmat(i,j)~=200
         fprintf(fid,'%d',Resmat(i,j));
      else
          fprintf(fid,'%s','w');
       end
   end
   fprintf(fid,'\t\t');
   fprintf(fid,'%d',Txmat(i,:));
   fprintf(fid,'\t');
   fprintf(fid,'%d',Oldmat(i,:));
   fprintf(fid,'\n');
end
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'%s','Time Invariant');
fprintf(fid,'\n');
[a2,b2]=size(C);
for q2=1:b2,
   fprintf(fid,'%d',C(1,q2));
end
fprintf(fid,'\n');
fprintf(fid,'\n');
[q4,w4]=size(Bmat);
for i1=1:q4,
   if Oldmat(i1)==2
      Ymat=[];
      Tmat=[];
      Ymat=[Ymat
           Bmat(i1,:)];
        Cmat=Bmat(i1,:);
```

```
        tx1=0;
       while tx1<1,
          for t1=1:q4,
             Cmat;
             Resmat(t1,:);
          if Cmat==Resmat(t1,:) & Oldmat(t1)== 0
             Cmat=Bmat(t1,:);
             Ymat=[Ymat
                Bmat(t1,:)];
             Tmat=[Tmat
                Txmat(t1)];
           break;
         end
          end
          if Cmat==m0
           tx1=1;
          break;
          end
           end
Road=Ymat;
K=max(Road);
Kmat=[Kmat
   K];
fprintf(fid,'%s','Road');
fprintf(fid,'\n');
[a3,b3]=size(Road);
for nq=1:a3,
   for mq=1:b3,
      fprintf(fid,'%d',Road(nq,mq));
   end
```

```
    fprintf(fid,'\n');
 end
fprintf(fid,'%s','K :');
fprintf(fid,'\n');
for q2=1:b3,
   fprintf(fid,'%d',K(1,q2));
end
fprintf(fid,'\n');
fprintf(fid,'\n');
end
end
 fprintf(fid,'\n');
 fprintf(fid,'\n');
 fprintf(fid,'\n');
 fprintf(fid,'\n');
end
fclose(fid);
Kx=Kmat;
```

EK-E

"cover.m", "Tinv.m", "Yöntem1.m", "Yöntem2.m" programlarının herbirinin çalıştırılması öncesinde analizi yapılacak Petri ağının aşağıda verilen formatta bir girdi dosyasıyla tanımlanması ve bu dosyanın Matlab komut penceresinde çağrılması gerekmektedir. Aşağıdaki girdi dosyası Örnek 4.1'deki Petri ağına aittir.

```
N= [1 0 0
    0 1 1
    0 1 0]

O= [0 1 1
    1 0 0
    1 0 0]

m0=[2 1 0]
```