BÖLÜM 1
SONLU ÖZDEVİNİRLER

1.1.Sonlu Özdevinir (FA)

Sonlu özdevinir (finite automata : FA) modeli, kesikli giriş ve çıkışları olan matematiksel bir modeldir. Bu model birçok donanım ve yazılım sisteminin matematiksel bir modelidir. Bu metin düzenleyici (text editör) ve derleyicilerin belirli kesimleri ile zaman uyumlu dizsel devreler (synchronous sequential circuits) sonlu özdevinir modeline uygun yazılım ve donanım sistemlerine örnek olarak verilebilir. Hatta sayısal bilgisayarlar bile bu model e uygun sistemler olarak görülebilir. Dolayısıyla bilgisayar bilimleri ve mühendisliği için sonlu özdevinir modeli, çok kullanılan önemli bir modeldir.

Sonlu özdevinir için bu kitapta, modelin İngilizce adının baş harflerinden oluşan ve yaygın biçimde kullanılan FA kısa adı kullanılacaktır.

Sonlu özdevinir modelinin birçok türü vardır. Sonlu özdevinirleri öncelikle:

A. Sonlu durumlu tanıyıcı (finite state recognizer)
B. Çıkış üreten özdevinir

modelleri olarak sınıflandırmak mümkündür. Bu sınıfların her birinin de kendi içinde altsınıfları vardır. Ancak kısaca “sonlu özdevinir” ya da FA denilince, temel model olan “deterministik sonlu durum tanıyıcı” modeli anlaşılır.

1.1.1.Deterministik Sonlu Özdevinir (DFA) Modeli

Temel modelde, deterministik sonlu özdevinir bir beşli olarak tanımlanır.

DFA=<Q, ∑, δ, q_0, F>

Q: Sonlu sayıda durum içeren Durumlar Kümesi

∑: Sonlu sayıda giriş simgesinden oluşan Giriş Alfabesi

q_0: Başlangıç durumu (q0 ϵ Q)

Başlangıç durumu durumlar kümesinin bir elemanı olduğuna göre Q boş olmayan (içinde en az q0 bulunan) bir kümedir.

F: Uç durumlar kümesi

Durumlar kümesinin bir altkümesidir. F ⊆ **Q**

δ: Durum geçiş işlevi (state transition function)

Durum geçiş işlevi ya da kısaca geçiş işlevi, (Q x ∑)'dan Q'ya bir eşleşme oluşturur.

Temel modelde geçiş işlevi ile her durum (durum, giriş simgesi) çiftine bir (durum) eşlenmektedir. Bu durumlardan ilki "şimdiki durum", ikincisi ise "sonraki durum" olarak adlandırılırsa, durum geçiş işlevi ile her (şimdiki durum, giriş simgesi) çiftine bir (sonraki durum) eşlendiği söylenebilir. Matematiksel olarak geçiş işlevi:

$$\delta(q_i, a) = q_j \qquad \forall q_i \in Q,\ a \in \Sigma \Rightarrow q_j \in Q$$

olarak tanımlanır.

Yukarıdaki tanımdan anlaşıldığı gibi, temel FA modeli deterministik bir modeldir. Bu model kısaca DFA (*deterministik finite automata*) olarak adlandırılır. Kısaca FA (*finite automata*) denildiğinde de DFA anlaşılır. Bu modele göre, FA q0 durumundan başlar ve uygulanan her giriş simgesi yeni bir duruma geçer. Her an FA'nın bulunduğu durum kesin olarak bellidir.

Örnek 1.1.

$M_{1.1}$ = DFA=<Q, ∑, δ, q_0, F>

Q = {q_1, q_2, q_3}

∑ = {0, 1}

F = {q_2}

δ: δ(q_0, 0) = q_0

δ(q_0, 1) = q_1

δ(q_1, 0) = q_0

δ(q_1, 1) = q_2

δ(q_2, 0) = q_2

δ(q_2, 1) = q_2

Geçiş Çizeneği

Örnek 1.1'de görüldüğü gibi, geçiş işlevinin matematiksel tanımı hem uzun, hem de anlaşılırlığı zor olan bir tanımdır. Bu nedenle geçiş işlevinin tanımı için genellikle "geçiş çizeneği (*transition diagram*)" olarak adlandırılan bir çizenek kullanılır.

Yönlü bir çizge olan geçiş çizeneğinde her durum için bir düğüm bulunur. Durum geçişleri ise yönlü yayar ile gösterilir ve yayların üzerine geçişe neden olan giriş simgesi yazılır. Geçiş çizeneğinde başlangıç durumu "→" ile, uç durumlar ise çift çember ile gösterilir. Örnek 1.1'deki FA için oluşturulan geçiş çizeneği Çizim 1.1'de görülmektedir.

Çizim 1.1. $M_{1.1}$'in Deterministik Geçiş Çizeneği

DFA'nın Tanıdığı Dizgiler Kümesi

Tanımlanan modele göre, DFA'nın girişine her seferinde bir giriş simgesi uygulanır. Her giriş simgesi DFA'nın bir sonraki duruma geçmesine neden olur. DFA, bulunduğu durumu yeni bir giriş simgesi uygulanana kadar korur. Uygulanan giriş simgelerinin sayısı ne kadar çok olursa olsun deterministik olduğu için model çalışır. Başlangıçta q0 durumunda bulunan

DFA, belirli sayıda giriş simgesinden oluşan bir dizinin (string) uygulamasından sonra belli bir duruma ulaşır. Uygulanan giriş dizgisi w, bu dizgi uygulandıktan sonra DFA'nın ulaştığı durumu da q1 ile gösterelim. $q0$,w ve $q1$ arasındaki ilişkiyi durum geçiş işlevini kullanarak ve bu işlevin anlamını biraz genişleterek aşağıdaki gibi gösterebiliriz:

$$\delta(q_0, w) = q_i$$

q0 durumundan başlayan ve w giriş dizgisinin uygulanmasından sonra q1 durumuna ulaşan DFA, eğer qi bir uç durum ise w giriş dizgisini tanır. Giriş alfabesi ∑={I1,I2,...,Ik} olan bir DFA'ya uygulanabilecek giriş dizgileri kümesi sonsuz bir kümedir. Giriş alfabesindeki simgelerden oluşan sonlu ya da sonsuz uzunluktaki her dizgi bu kümede yer alır. Giriş alfabesindeki simgelerden oluşan dizgilerin bir kısmı DFA tarafından tanınan, diğer kısmı ise tanınmayan dizgilerdir. Bu aşamada, bir sonlu özdevinir (M) tarafından tanınan dizgiler kümesini aşağıdaki gibi tanımlamak mümkündür.

$$T(M) = \{w \mid \delta(q_0, w) = q_i \in F\}$$

Buna göre DFA'ya uygulanabilecek dizgiler kümesi, DFA tarafından tanınan ve tanınmayan olmak üzere iki altkümeye ayrılmaktadır. DFA'nın tanıdığı küme, DFA'yı başlangıç durumundan bir uç duruma götüren dizgilerin kümesidir. Örnek 1.1'de tanımlanan DFA'yı (M1.1) incelersek, örneğin 0101 dizgisinin M1.1 tarafından tanınmadığını; buna karşılık 0110 dizgisinin M1.1 tarafından tanındığını görürüz. Çünkü q0 durumundan başlaya M1.1, 0101 uygulandıktan sonra q1 durumuna ulaşır ve q1 bir uç durum değildir. Buna karşılık 0110 dizgisi M1.1'i başlangıç durumundan q2 durumuna götürür ve q2 bir uç durumdur. M1.1'in tanıdığı dizgiler kümesi sonsuz bir kümedir. Bu kümedeki dizgilerden birkaç örnek aşağıda gösterilmiştir.

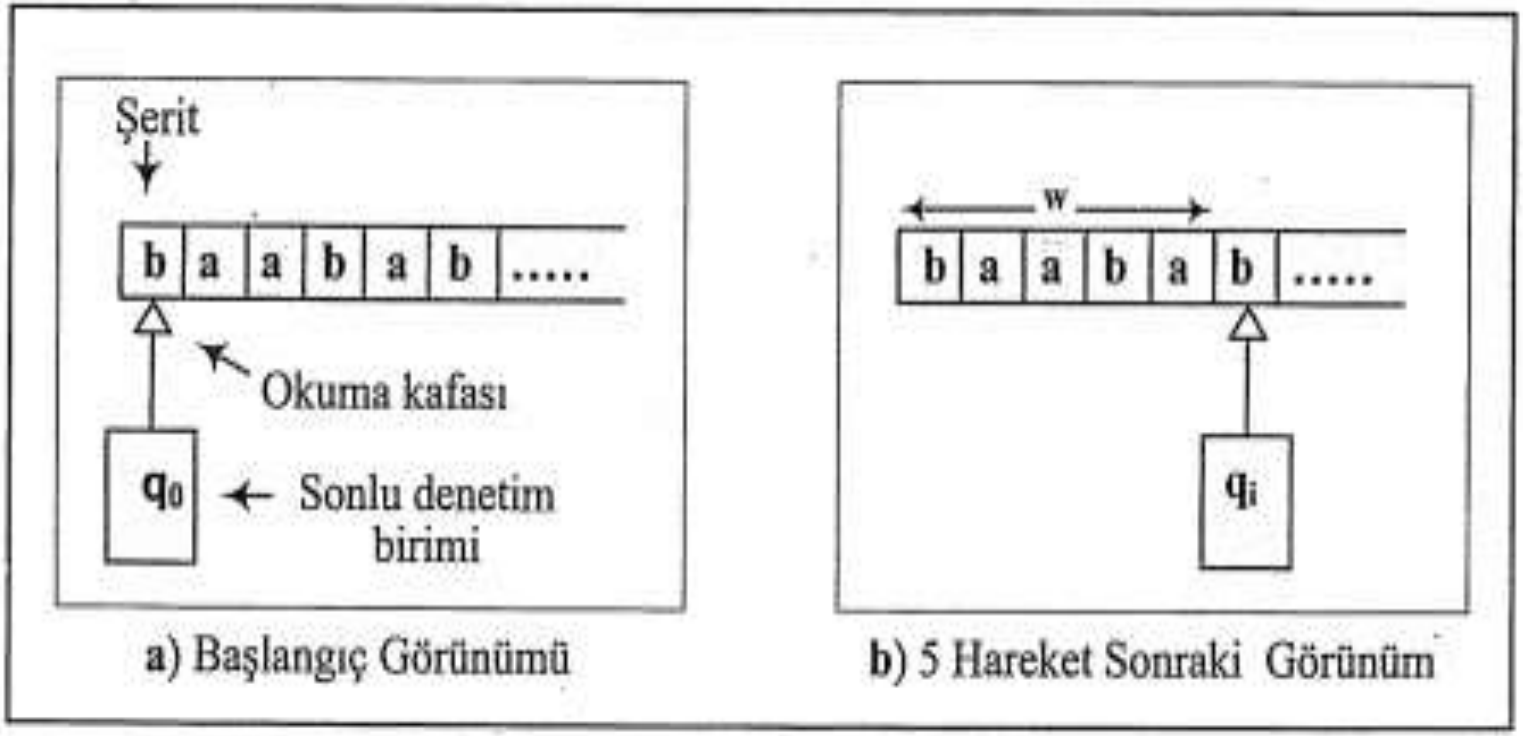
$$T(M_{1.1}) = \{11, 011, 110, 0110, 0110, 01011, \ldots\}$$

Çizim 1.1'deki geçiş çizeneği incelendiğinde, bir dizginin M1.1 tarafından tanınabilmesi için dizgide art arda iki tane 1 bulunması gerekli ve yeterli olduğu görülür. Buna göre T(M1.1) sözlü olarak,{0,1} alfabesinde, içinde 11 alt dizgisi bulunan dizgiler kümesi olarak tanımlanabilir.

DFA'nın Şeritli Makine Modeli

Tanımlandığı biçimiyle deterministik sonlu özdevinir modeli matematiksel bir modeldir. Deterministik bir sonlu özdevinir oluşturmak için bir durumlar kümesi ile bir giriş alfabesi oluşturmak, durumlardan birini başlangıç durumu olarak seçmek, durumların bir kesimini uç (tanıyan) durumlar olarak nitelemek ve bir geçiş işlevi tanımlamak yeterlidir. Oluşturulan DFA soyut bir makine olarak görülebilir ve kısaca "makine" olarak adlandırılır. Ancak "makine" adlandırmasının tamamen simgesel olduğu, oluşturulan sonlu özdevinirin makine ile hiç benzerliği olmayan bir şey de olabileceği unutulmamalıdır.

Soyut bir makine olan deterministik sonlu özdevinirin daha kolay anlaşılması için, somut bir şeritli makine modeli düşünülebilir. Bu amaçla, DFA için kullanılan somut makine modeli, bilinen bileşenlerden oluşan ve Çizim 1.2'de görülen bir modeldir.



Çizim 1.2. DFA'nın Şeritli Makine Modeli

Çizim 1.2. DFA'nın Şeritli Makine Modeli Başlangıç Görünümü

Çizim 1.2. DFA'nın Şeritli makine Modeli 5 Hareket Sonraki Görünüm

Çizim 1.2.a'da görüldüğü gibi DFA'nın makine modeli 3 elemandan oluşmaktadır.

- Hücrelerden oluşan ve her hücresinde bir giriş simgesi bulunan bir mıknatıslı şerit. Yalnız okunabilen şeridin sağ ucu sonsuzdur. Başlangıçta şerit üzerinde giriş simgelerinden oluşan bir dizgi kayıtlıdır.
- Bir sonlu denetim birimi(SDB).SDB'nin sonlu sayıda durumu vardır. Bu durumlardan biri başlangıç durumudur ve SDB başlangıçta bu durumda bulunur.
- Bir okuma kafası. Şeridin okunması soldan sağa doğru tek yönlü gerçekleşir. Belirli bir anda okuma kafası şeridin hücrelerinden biri üzerinde bulunur ve üzerinde bulunduğu hücrede kayıtlı simgeyi okuyabilir.

Çizim 1.2'deki modele göre, DFA (makine) aşağıda açıklanan biçimde çalışır:

- Başlangıçta şerit üzerinde bir giriş dizgisi kayıtlıdır ve okuma kafası şeridin ilk (en soldaki) hücresi üzerindedir.
- Makinenin her hareketinde aşağıdaki işlemler yapılır:
    1. Şeritten bir simge okunur.
    2. Okuma kafası bir sağdaki hücreye geçer.
    3. Sonlu denetim birimi bir sonraki duruma geçer.
- Belirli bir sayıda hareket sonunda okuma kafası belirli bir hücrenin üzerinde, sonlu denetim birimi ise belirli bir durumda (qi) bulunur. Okuma kafasının üzerinde bulunduğu hücrenin solundaki hücrelerde yer alan dizgiyi w olarak adlandıralım. Eğer qi bir uç durumsa makine w'yi tanır(Çizim 1.2.b).

1.1.2.Deterministik Olmayan Sonlu Özdevinir (NFA) Modeli

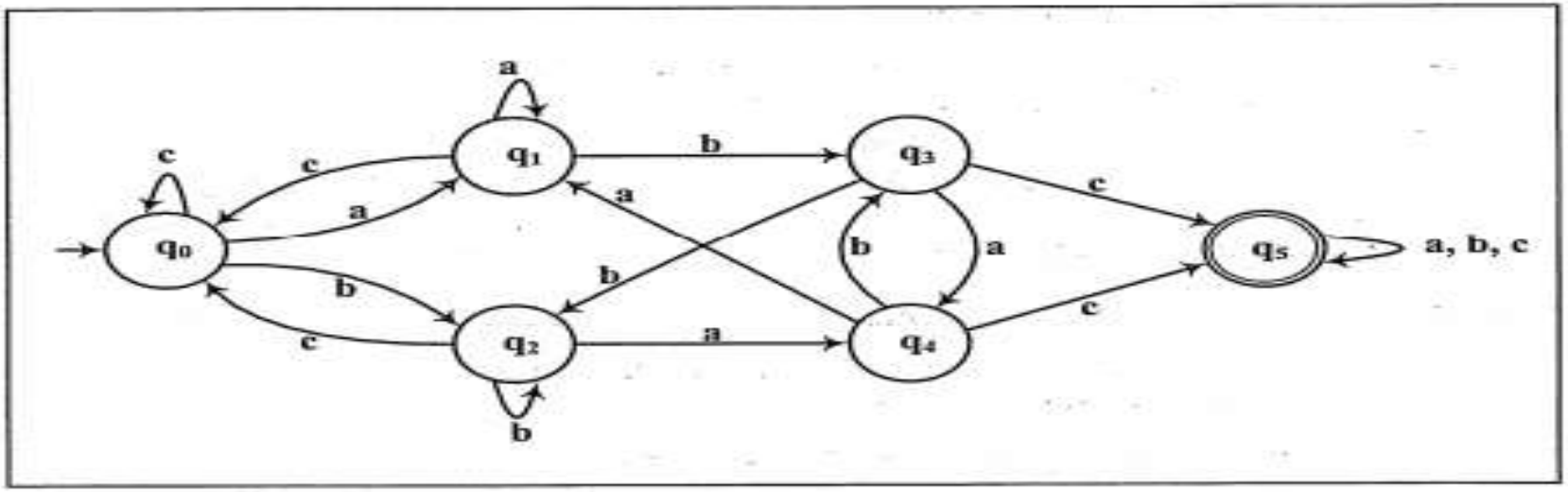
Sözlü olarak ya da bir düzgün deyimle (bkz 2.2.) tanımlanan belirli bir dizgiler kümesini tanıyan DFA'nın deterministik geçiş çizeneğinin oluşturulması oldukça zor olabilir.

Örnek 1.2. "{a,b,c} alfabesinde, içinde abc ve bac alt dizgilerinden en az biri, en az bir kez bulunan dizgiler kümesi'ni düşünelim ve bu kümeyi tanıyan DFA'yı $M_{1.2}$ olarak adlandıralım.

$M_{1.2}$'nin tanıdığı dizgilerden birkaç örnek aşağıda gösterilmiştir.

$T(M_{1.2})$={abc, bac, cbabcb, bccaabcbac,…}

$M_{1.2}$'nin deterministik geçiş çizeneği ise Çizim 1.3'de görülmektedir. $M_{1.2}$ 'nin tanıdığı kümenin sözlü tanımı oldukça kolay olmasına karşın, geçiş çizeneğinin oldukça karmaşık olduğu görülmektedir. Üstelik sözlü tanımdan hareketle bu çizeneğin elde edilmesini sağlayan sistematik bir yöntem de yoktur.



Çizim 1.3. $M_{1,2}$ 'nin Deterministik Geçiş Çizeneği

Deterministik modelin kullanım güçlüğü nedeniyle, kullanımı daha kolay ve daha esnek bir model olan deterministik olmayan(*non deterministic*) model geliştirilmiştir. Deterministik olmayan sonlu özdevinir (*non deterministic finite automata* : NFA) aşağıdaki gibi tanımlanır:

$$NFA = <Q, \sum, \delta, q_0, F>$$

Bu tanımda Q,∑,q0 ve F'nin anlamları deterministik model tanımındakilerle aynıdır. Deterministik modelde olduğu gibi deterministik olmayan model de Q durumlar kümesini, ∑ giriş alfabesini, q0 başlangıç durumunu, F ise uç durumlar kümesini göstermektedir. İki model arasındaki tek fark geçiş işlevinin tanımındadır. Deterministik modelde geçiş işlevi:

- (Q x ∑)'dan Q'ya

bir eşleşme olarak tanımlanırken, deterministik olmayan modelde geçiş işlevi:

- (Q x ∑)'dan Q'nun altkümelerine

bir eşleşme olarak tanımlanır. Buna göre deterministik modelde her durumdan her giriş simgesi ile bir ve yalnız bir duruma geçilirken, deterministik olmayan modelde bir durumdan bir giriş simgesi ile geçilebilecek durum sayısı sıfır, bir ya da birçok olabilmektedir.

Örnek 1.3. $M_{1.3} = <Q, \sum, \delta, q_0, F>$

$Q = \{ q_0, q_1, q_2, q_3\}$

$\sum = \{0, 1\}$

$F = \{ q_2\}$

$\delta = \delta\{q_0, 0\} = \{ q_0, q_1\}$

$\delta\{q_0, 1\} = \{ q_0, q_2\}$

$\delta\{q_1, 0\} = \{ q_3\}$

$\delta\{q_1, 1\} = \{ \} = \phi$

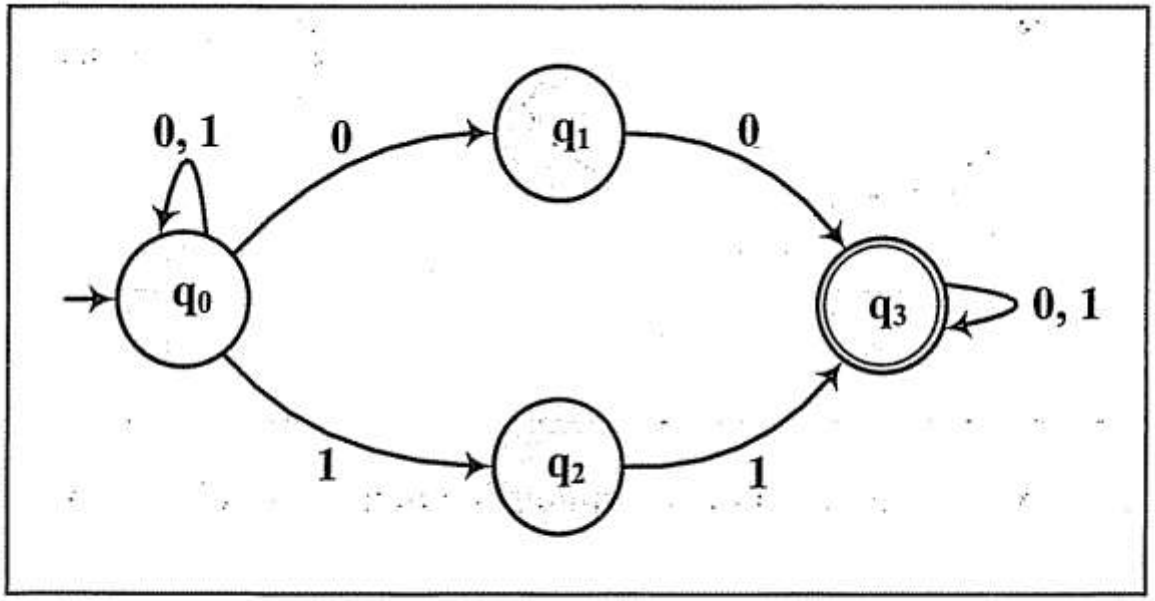
$\delta\{q_2, 0\} = \phi$

$\delta\{q_2, 1\} = \{q_3\}$

$\delta\{q_3, 0\} = \{q_3\}$

$\delta\{q_3, 1\} = \{q_3\}$

$M_{1.3}$'ün tanımında görüldüğü gibi, deterministik olmayan modelde bazı durumlarda bazı giriş simgeleri ile birden çok duruma geçilebilmektedir. Bazı durumlardan bazı giriş simgeleri ile de hiçbir duruma geçilememektedir. Bu nedenle, deterministik modelde, başlangıç durumu ve uygulanan giriş dizgisi bilindiğinde makinanın hangi durumda bulanacağı kesin olarak bellidir. Buna karşılık, deterministik olmayan modelde başlangıç durumu ve uygulanan giriş dizgisi bilinen makinenin son durumu belirsiz olabilir. Örneğin başlangıç durumu q0 olan ve $M_{1.3}$'e w=000 giriş dizgisi uygulandığında, makinenin son durumu q0, q1 ve q3 durumlarından herhangi biri olabilir.



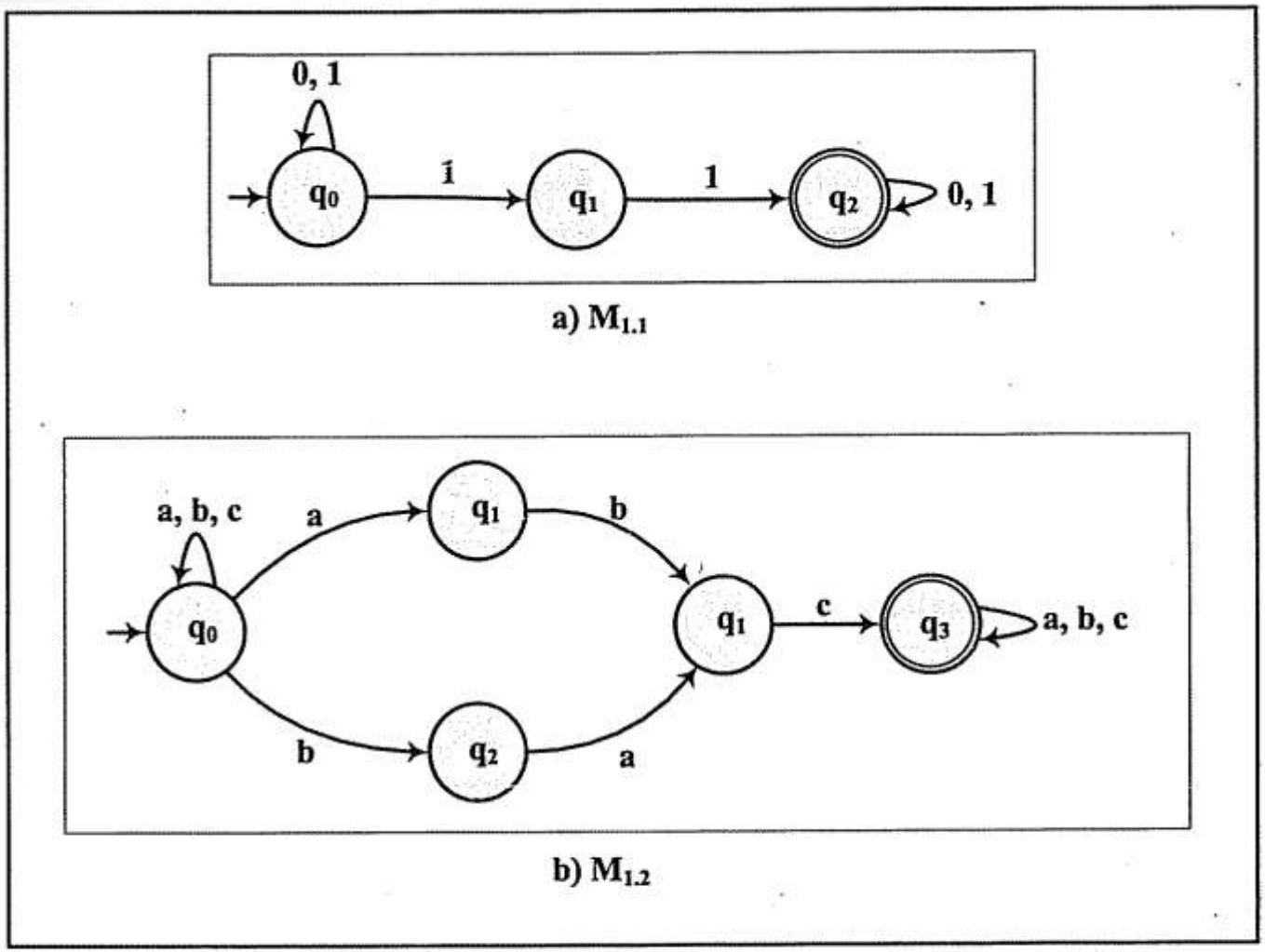
Çizim 1.4. $M_{1.3}$'ün Deterministik Olmayan Geçiş Çizeneği

Çizim 1.4.M1.3'nin Deterministik Olmayan Geçiş Çizeneği

Deterministik olmayan bir sonlu özdevinirin(NFA) bir giriş dizgisini tanıması için, geçiş çizeneğinde, başlangıç durumundan başlayarak bu giriş dizgisi ile bir uç duruma ulaşan bir yol bulunabilmesi gerekir. Örneğin $M_{1.3}$, w=10001 giriş dizgisini tanır. Çünkü bu giriş dizgisine karşı gelen ve q_0'dan başlayarak tek uç durum olan q_3'te son bulan bir yol

$$(q_0 \to q_0 \to q_1 \to q_3 \to q_3 \to q_3 \text{ ya da } q_0 \to q_0 \to q_0 \to q_1 \to q_3 \to q_3)$$

bulunabilmektedir. Buna karşılık $M_{1.3}$, w=010 giriş dizgisini tanımaz. Çünkü q_0 ve q_3 durumları arasında bu giriş dizgisine karşı gelen bir yol bulmak mümkün değildir. Deterministik olmayan geçiş çizeneklerinin oluşturulması ve okunması deterministik çizeneklere göre daha kolaydır. Örneğin Çizim 1.4 incelendiğinde, $M_{1.3}$'ün tanıdığı kümenin "{0,1} alfabesinde, içinde 00 ve 11 alt dizgilerinden en az biri, en az bir kez bulunan dizgiler kümesi" olduğu kolaylıkla bulunabilir. Oysa Çizim 1.3 incelenerek, $M_{1.2}$'nin tanıdığı kümeyi bulmak, başka bir deyişle Çizim 1.3'teki deterministik geçiş çizeneğinin oluşturmak hiç de kolay değildir. Bir karşılaştırma olanağı sağlamak için Çizim 1.5'te $M_{1.1}$ ve $M_{1.2}$'nin deterministik olmayan geçiş çizenekleri verilmiştir.



Çizim 1.5. $M_{1.1}$ ve $M_{1.2}$ 'nin Deterministik Olmayan Geçiş Çizenekleri

Çizim 1.5.a. $M_{1.1}$ ve $M_{1.2}$'nin Deterministik Olmayan Geçiş Çizeneği

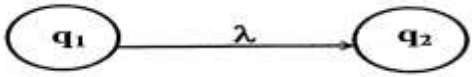
Çizim 1.5.b. $M_{1.1}$ ve $M_{1.2}$'nin Deterministik Olmayan Geçiş Çizeneği

1.1.3.Lambda(λ) Geçişi

Sonlu özdevinirlerde durum geçişleri, giriş simgelerinin işlenmesi ile gerçekleşir. Makine belli bir durumdayken, bir geçiş simgesinin uygulanması makinenin bir sonraki duruma geçmesine neden olur. Geçiş işleviyle tanımlanan "sonraki durum" deterministik modelde tek bir durumdur. Deterministik olmayan modelde ise "sonraki durum" durumlar kümesinin bir altkümesi olup bu altkümede sıfır, bir ya da birçok durum bulunabilir. Şimdiye kadar yapılan tanımlara göre, DFA ya da NFA modeline uygun bir makine, giriş simgesi uygulanmadığı sürece bulunduğu durumu korur. Lambda geçişi ile deterministik olmayan modelle genişletilen ve kullanımı kolaylaştırılan sonlu özdevinir tanımı daha da genişletilir. Lambda ve lambda geçişi soyut kavramlardır. Lambda bir boş simge olarak düşünülebilir. Lambda geçişi ise, hiçbir giriş simgesi uygulanmadan (ya da işlenmeden) gerçekleşen durum geçişine karşı gelir. Bir makinenin q1 ve q2 durumları arasında

$$\delta(q_1, \lambda) = q_2$$

biçiminde tanımlanan λ-geçişi varsa, q1 durumunda bulunan makinenin, hiçbir giriş simgesi işlenmeden, kendiliğinden q2 durumuna geçebileceği anlaşılır.



Lambda geçişi, modelin esnekliğini arttıran; geçiş çizeneklerinin daha kolay oluşturulmasını ve okunmasını sağlayan soyut bir kavramdır. Diğer geçiler gibi, lambda geçişi de tek yönlüdür. q1 ve q2 durumları arasındaki, q1'den q2'ye λ-geçişi, hiçbir giriş simgesi uygulanmadan makinenin q1 durumundan q2 durumuna geçebileceğini gösterir. Buna göre, q1 durumunda bulunan makinenin aynı zamanda q2 durumunda da bulunduğu anlamı çıkar. Ancak bunun tersi doğru değildir.

q1 ve q2 durumları arasında, q1'den q2'ye λ-geçişi varsa işlevsel olarak:

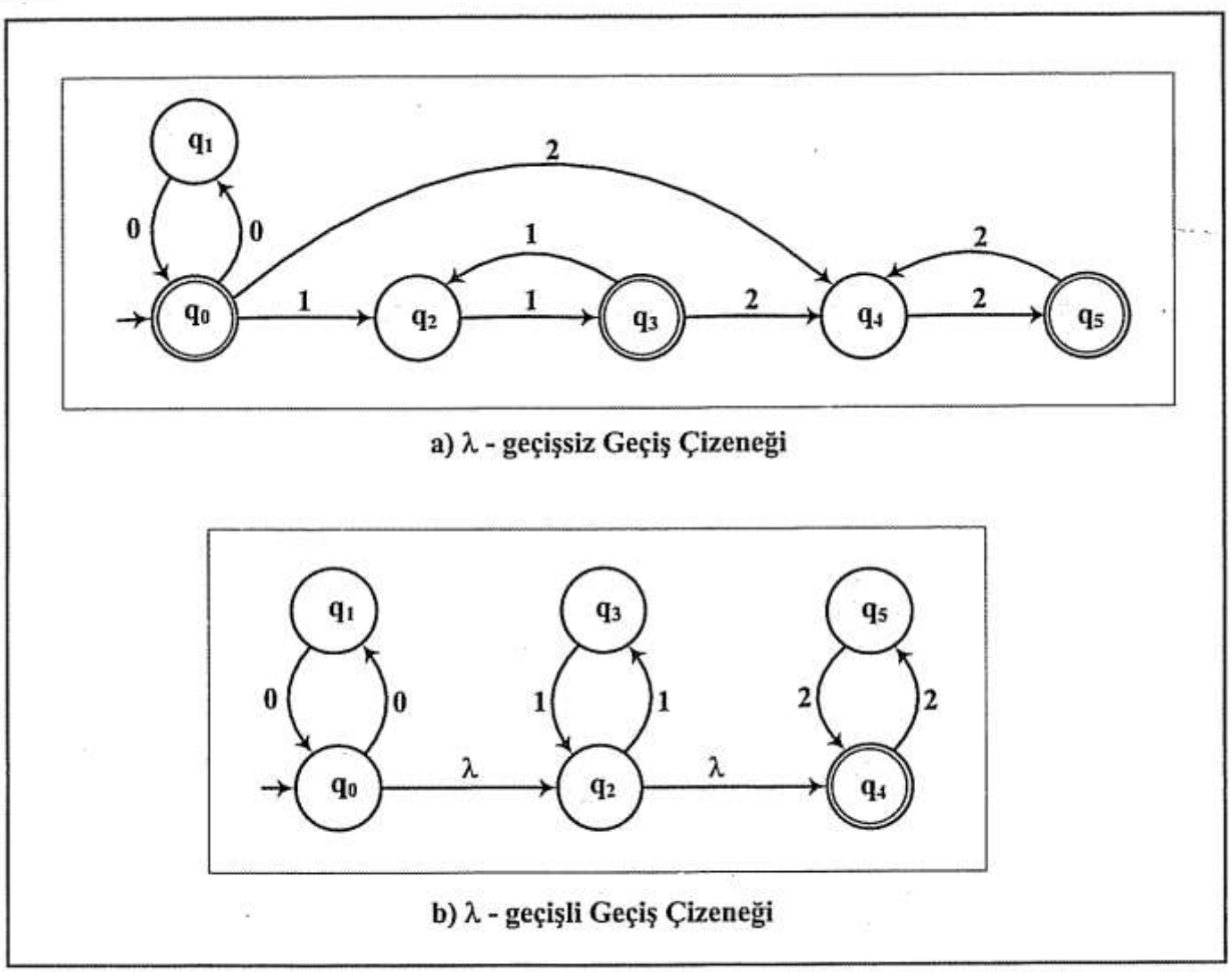
- eğer q1 başlangıç durumu ise q2 de başlangıç durumu,
- eğer q2 bir uç durum ise q1 de bir uç durum

niteliği kazanır.

Örnek 1.4. Lambda geçişlerinin sonlu özdevinir modeline sağladığı esnekliği göstermek için, tanıdığı küme {0,1,2} alfabesinde:

$$T(M_{1.4}) = \{0^{2n}\,1^{2m}\,2^{2k} \quad | \quad n \geq 0,\ m \geq 0,\ k \geq 0\}$$

biçiminde tanımlanan makineyi düşünelim. M1.4'ün, λ-geçişi kullanmadan oluşturulan geçiş çizeneği Çizim 1.6.a'da, λ-geçişi kullanılarak oluşturulan geçiş çizeneği ise Çizim 1.6.b'de görülmektedir. λ-geçişsiz ve λ-geçişli geçiş çizenekleri incelendiğinde, λ-geçişlerinin sonlu özdevinir modeline kazandırdığı esneklik ve kullanım kolaylığı açık biçimde görülmektedir.



Çizim 1.6. $M_{1.4}$ için Geçiş Çizenekleri : a) λ-Geçişsiz b) λ- Geçişli

Çizim 1.6.a. $M_{1.4}$ için Geçiş Çizenekleri- λ Geçişsiz

Çizim 1.6.b. $M_{1.4}$ için Geçiş Çizenekleri- λ Geçişsiz

λ-Geçişsiz Eşdeğer Geçiş Çizeneğinin Bulunması

λ-geçişlerinin kullanımı, geçiş çizeneklerinin kullanımını kolaylaştırır ancak gücünü arttırmaz. Başka bir deyişle λ-geçişleri kullanılarak oluşturulan her geçiş çizeneğine denk, λ-geçişsiz bir geçiş çizeneği bulunabilir. İki geçiş çizeneğinin denkliği, tanıdıkları kümenin eşitliği anlamını taşımaktadır. Buna göre, λ-geçişli bir geçiş çizeneği tarafından tanınan, ancak λ-geçişsiz hiçbir geçiş çizeneği tarafından tanınmayan hiçbir dizgiler kümesi yoktur. Bu da, geçiş çizeneklerinde λ-geçişi kullanmanın, NFA modelinin ifade gücünde hiçbir değişiklik yapmadığını gösterir.

λ-geçişli bir geçiş çizeneği verildiğinde, λ-geçişlerini tek tek yok ederek eş değer bir geçiş çizeneği elde etmek mümkündür. Bu kapsamda, q1 ve q2 durumları arasında q1'den q2'ye λ-geçişi varsa:

- q2 durumundan başlayan her durum geçişine (√δ(q2,a)=qk) karşılık q1 durumundan başlayan ve aynı giriş simgesi ile aynı duruma ulaşan bir durum geçişi (δ(q1,a)=qk) eklendikten,

- eğer q1 durumu başlangıç durumu ise, q2 durumu da başlangıç durumu yapıldıktan ve

- eğer q2 durumu bir uç durum ise, q1 durumu da uç durum yapıldıktan sonra , λ-geçişi silinebilir.

Örnek 1.5. $M_{1.5}$ makinesi {a.b.c} alfabesinde, sıfır ya da iki tane a ile, ya da çift sayıda b (0,2,4,… tane b) ile başlayıp cc ile biten dizgiler kümesini tanıyan makine olsun. $M_{1.5}$'in tanıdığı kümedeki dizgilerden birkaç örnek aşağıda görülmektedir.

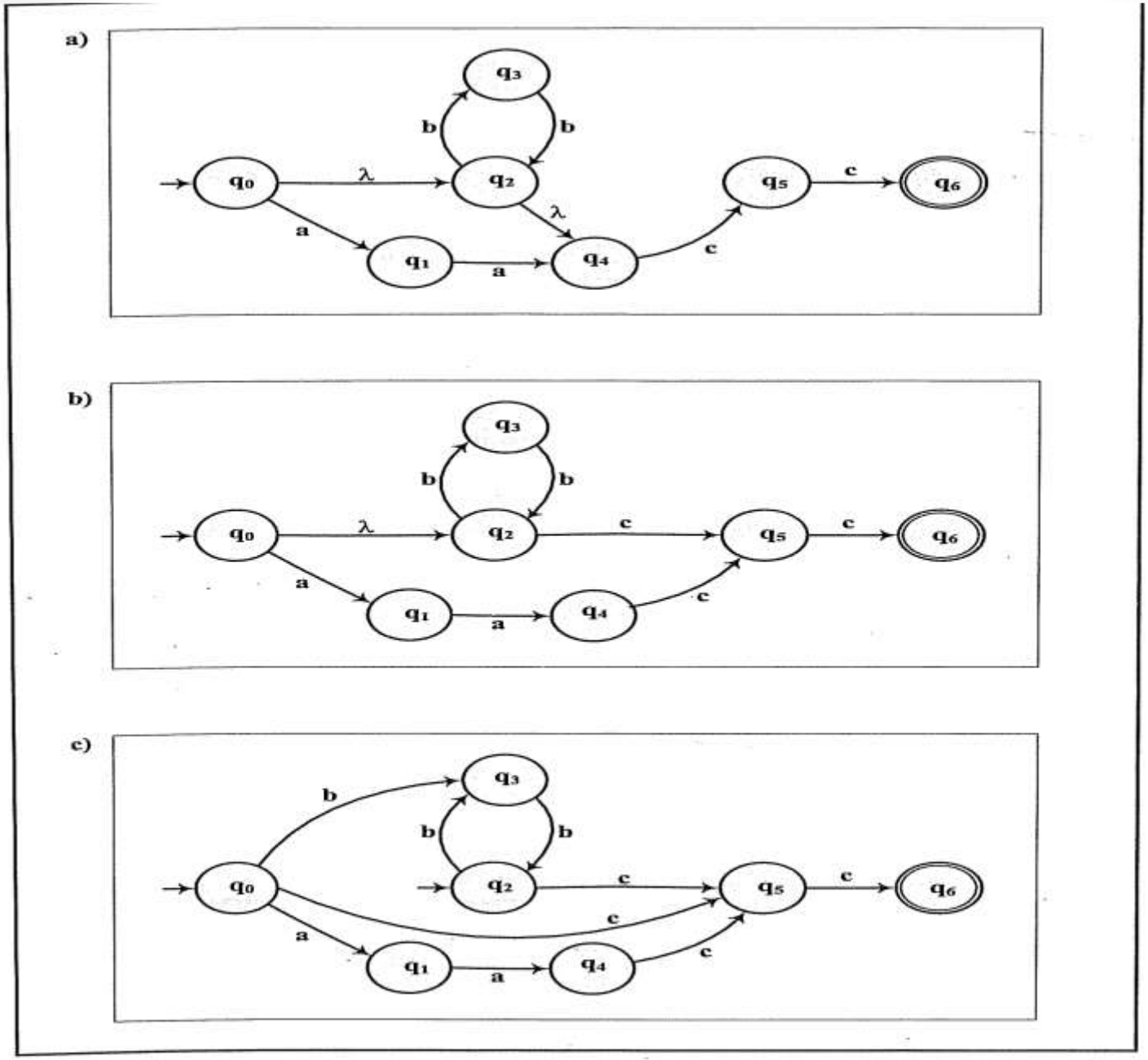
$$\mathbf{T(M_{1.5})=\{cc,aacc,bbcc,bbbbcc,bbbbbbcc,\ldots\}}$$

$M_{1.5}$ için oluşturulan λ-geçişli geçiş çizeneği Çizim 1.7.a'da görülmektedir. bu çizeneğe eş değer, λ-geçişsiz geçiş çizeneğini elde edebilmek için, önce q_2 ile q_4 arasındaki, sonra da q_0 ile q_2 arasındaki λ-geçişleri yok edilir. q_2 ile q_4 arasındaki λ-geçişini yok edebilmek için q_2 ile q_5 arasına c-geçişi eklenir ve Çizim 1.7.b'deki geçiş çizeneği elde edilir. q0 ile q_2 arasındaki λ-geçişini yok edebilmek için ise q_0 ile q_3 arasına b-geçişi, q_0 le q_5 arasına c-geçişi eklenip q_2 durumu başlangıç durumu yapılır ve Çizim 1.7.c'deki λ-geçişsiz geçiş çizeneği elde edilir.

Eğer λ-geçişleri yok edilerek geçiş çizeneğinde zincir yapısında birden çok λ-geçişi varsa, yok etme işlemini zincirin ucundan başlatarak geriye doğru sürdürmekte yarar vardır. Örneğin q_1 ile q_2; q_2 ile q_3; q_3 ile de q_4 arasında üç λ-geçişi varsa:

- önce q_3 ile q_4 arasındaki,
- sonra q_2 ile q_3 arasındaki,
- son olarak da q_1 ile q_2 arasındaki

λ-geçişi yok edilir.



Çizim 1.7. Geçiş Çizeneğinde λ-Geçişlerinin Yok Edilmesi

Çizim 1.7.a

Çizim 1.7.b

Çizim 1.7.c

1.1.4.Deterministik ve Deterministik Olmayan Sonlu Özdevinir Modellerinin Denkliği

Acaba deterministik ve deterministik olmayan özdevinir modelleri denk midir? Eğer her iki model ile tanınabilen kümeler sınıfı aynı ise bu iki model denktir. Yok Eğer modellerden biri ile tanınabilen ancak diğeri ile tanınamayan kümeler varsa bu iki model denk değildir. Tanımlarına göre deterministik Olmayan model deterministik modeli de kapsadığına başka bir deyişle her DFA aynı zamanda bir NFA olduğuna göre, deterministik bir FA tarafından tanınan ancak hiçbir NFA tarafından tanınmayan bir küm düşünülemez. Ancak NFA Tarafından tanınan her küme için bu kümeyi tanıyan bir DFA Bulmak mümkün müdür? Eğer bu sorunun yanıtı Evet ise deterministik ve deterministik olmayan modellerin denk olduğu söylenebilir.

Yukarıdaki sorunun yanıtını bir başka soruya yanıt arayarak bulmaya çalışalım. Bir dizgi ve bir DFA verildiğinde geçiş çizeneğini kullanarak bu dizginin bu DFA tarafından tanınıp tanımadığını bulmak kolaydır ancak bir dizgi ve bir NFA verildiğinde geçiş çizeneğini kullanarak bu dizginin bu NFA tarafından tanınıp tanınmadığı bulmak kolay değildir.

Örnek 1.6. $M_{1.6} = <Q, \sum, \delta, q_0, F>$

$Q = \{A, B, C\}$

$\sum = \{0, 1\}$

$q_0 = A$

$F = \{C\}$

δ: $\delta(A, 0) = \{A\}$

$\delta(A, 1) = \{B, C\}$

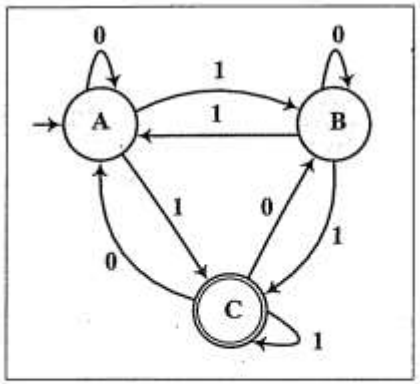
$\delta(B, 0) = \{B\}$

$\delta(B, 1) = \{A, C\}$

$\delta(C, 0) = \{A, B\}$

$\delta(C, 1) = \{C\}$

Örnek olarak $M_{1.6}$'yı düşünelim. Deterministik olmayan modele göre tanımlanan bu makinenin geçiş çizeneği Çizim 1.8.a'da görülmektedir. Başlangıç durumu A, tek uç durum da C olduğuna göre, verilen bir dizginin $M_{1.6}$ tarafından tanınabilmesi için A ve C durumları arasından, bu dizgiye karşı gelen bir yolun bulunabilmesi gerekir. Bu işlemin sistemli bir şekilde yapılmasını sağlayacak bir yöntem bulmaya çalışalım. Bu amaçla Çizim 1.8.b'de görülen geçiş çizelgesinden yararlanabiliriz. Geçiş çizelgesi, geçiş çizeneği ile aynı bilgiyi içeren ve makinenin her durumundan, her giriş simgesi ile hangi durumlara gidebileceğini gösteren bir çizelgedir. Başka bir deyişle geçiş çizelgesi her durumun her giriş simgesine karşı gelen ardıllarını kolaylıkla bulmayı



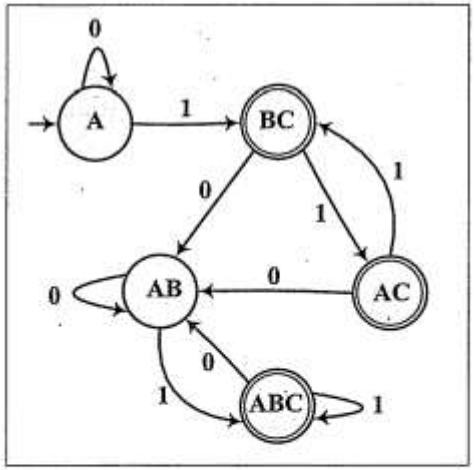
a) Deterministik Olmayan Geçiş Çizeneği

	0	1
→A	A	BC
B	B	AC
(C)	AB	C

b) Geçiş Çizelgesi

	0	1
→A	A	BC
(BC)	AB	AC
AB	AB	ABC
(AC)	AB	BC
(ABC)	AB	ABC

c) Başlangıç Durumunun Ardılları Çizelgesi



d) Deterministik Geçiş Çizeneği

Çizim 1.8. NFA'ya Denk DFA'nın Bulunması

Çizim 1.8.a

Çizim 1.8.c

sağlar. Ancak verilen bir w giriş dizgisini $M_{1.6}$ tarafından tanınabilmesi için başlangıç durumu olan A'nın w-ardılları arasında tek uç durum olan C'nin yer alması gerekir. Bu amaçla, Çizim 1.8.c'de görülen ve başlangıç durumunun tüm ardıllarını içeren çizelgeyi hazırlayabiliriz. Bir w giriş dizgisi verildiğinde bu ardıl çizelgesi yardımıyla, w'nin M1.6 tarafından tanınıp tanınmadığı kolaylıkla bulunabilir. Örneğin w=011 ise A'nın 011-ardılı AC olduğu ve AC bir uç durum içerdiği için de 011 dizgisinin M1.6 tarafından tanındığı; buna karşılık w=110 ise A'nın 110-ardılının AB olduğu ve AB içinde bir uç durum bulunmadığı için de 110 dizgisinin $M_{1.6}$ tarafından tanınmadığı kolaylıkla bulunabilir.

$M_{1.6}$'da başlangıç durumunun ardıllarını gösteren Çizim 1.8.c'deki çizelge ile deterministik bir özdevinir tanımlanmaktadır. Bu özdeviniri $Md_{1.6}$ olarak adlandıralım. $Md_{1.6}$'nın biçimsel tanımı aşağıdaki gibidir.

$M_{D1.6} = \langle Q, \sum, \delta, q_0, F \rangle$

$Q = \{A, BC, AB, AC, ABC\}$

$\sum = \{0, 1\}$

$q_0 = A$

$F = \{BC, AC, ABC\}$

δ: $\delta(A, 0) = A$

$\delta(A, 1) = B,C$

$\delta(BC, 0) = AB$

$\delta(BC, 1) = AC$

$\delta(AB, 0) = AB$

$\delta(AB, 1) = ABC$

$\delta(AC, 0) = AB$

$\delta(AC, 1) = BC$

$\delta(ABC, 0) = AB$

$\delta(ABC, 1) = ABC$

$M_{D1.6}$'nın geçiş çizeneği Çizim 1.8.d'de görülmektedir. $M_{D1.6}$ verilen bir w giriş dizgisinin $M_{1.6}$ tarafından tanınıp tanınmadığını bulmak üzere geliştirildi. Dolayısıyla $M_{1.6}$ ve $M_{D1.6'nın}$ tanıdığı dizgiler kümesinin aynı olduğu ve bu iki makinenin denk olduğu açıktır. Başka bir deyişle $M_{D1.6}$ deterministik olmayan M1.6 makinesiyle aynı kümeyi tanıyan dolayısıyla $M_{1.6}$'ya denk deterministik bir makinedir. Bir kez bulduktan sonra deterministik makinenin durumlarına A,B,C,… ya q_0,q_1,q_2 gibi adlar vererek durumları yeniden adlandırmak mümkündür.

1.1.5.İki Yönlü Sonlu Özdevinir Modeli

Sonlu özdevinirlerin temel modeli olan DFA modeli tek yönlü bir modeldri. Çizim 1.2'de soyut bir makine olarak sunulan bu modelde, okuma şeridi soldansağa doğru tek yönlü işlenir. DFA'nın her hareketinde, şeritten bir simge okunur ve okuma kafası bir sağdaki hücreye geçer. Tek yönlü modelde okuma kafası şerit üzerinde sola doğru hareket edemez.

İki yönlü sonlu özdevinir modelinde ise okuma kafasının hem sağa hem de sola doğru hareket etmesi mümkündür. Makinenin her hareketinde okuma kafası, üzerinde bulunduğu hücredeki simgeyi okuduktan sonra ya bir sağdaki ya da bir soldaki hücreye geçer.

İki yönlü sonlu özdevinirlerin yalnız deterministik modeli incelenecek ve bu model 2DFA olarak adlandırılacaktır. 2DFA bir beşli olarak aşağıdaki gibi tanımlanır.

2DFA=<Q,∑,δ,q0,F>

2DFA'nın tanımında yer alan durumlar kümesi (Q), giriş alfabesi (∑), başlangıç durumu (q0) ve uç durumların (F) anlamları, DFA tanımındakilerle aynıdır. (bkz 1.1.1). DFA ve 2DFA arasındaki tek fark geçiş işlevinin tanımındadır. 2DFA modelinde geçiş işlevi:

(Q x ∑)'dan Q x {R,L}'ye bir eşleme

olarak tanımlanır. Bu tanımda R (Right) ve L(Left), okuma kafasının bir sağ mı yoksa bir sola mı geçeceğini gösterir.

Örnek 1.7. $M_{1.7} = <Q, \sum, \delta, q_0, F>$

$Q = \{q_0, q_1, q_2\}$

$\sum = \{0, 1\}$

$F = \{q_0, q_1, q_2\}$

δ: $\delta(q_0, 0) = \{q_0, R\}$

$\delta(q_0, 1) = \{q_1, R\}$

$\delta(q_1, 0) = \{q_1, R\}$

$\delta(q_1, 1) = \{q_2, L\}$

$\delta(q_2, 0) = \{q_0, R\}$

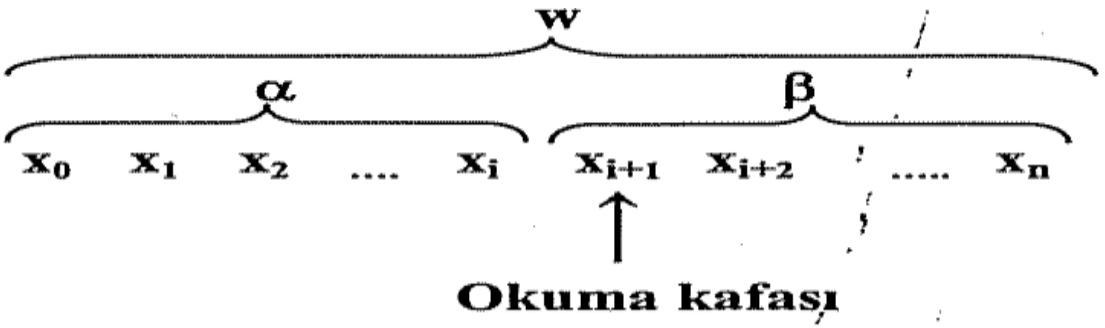
$\delta(q_2, 1) = \{q_2, L\}$

Örnek 1.7'deki $M_{1.7}$ makinesi, {0,1} alfabesinde içinde 11 alt dizgisi bulunmayan dizgileri tanımlayan bir 2DFA'dır. Tüm durumlar uç durum olduğuna göre, bir dizginin $M_{1.7}$ tarafından tanınabilmesi için sonlu sayıda hareket sonunda tüm dizginin işlenmesi ve okuma kafasının dizginin sağına geçmesi yeterlidir. Tek yönlü modelde her zamana sağlanan bu koşul iki yönlü özdevinirlerde her zaman sağlanamaz. İki nedenle okuma kafası sonlu hareket sonunda dizginin sağına ulaşamaz. Bu nedenlerden birincisi okuma kafasının hareketlerinin bir döngü oluşturması, ikinci ise şeridin ilk hücresi üzerinde bulunan okuma kafasının sola ilerlemesini gerektiren bir hareketle karşılaşılmasıdır.

Bir dizginin 2DFA tarafından tanınıp tanınmadığını bulmak pek kolay değildir. Bunun için şeritli makine modeli üzerinde okuma kafasının hareketlerinin izlenmesi gerekir. Ancak hareketler iki yönlü olduğu ve aynı hücre üzerinden okuma kafası defalarca geçebildiği için bu işlem oldukça uzun ve güç olabilir. Diğer bir yöntem ise, şeritli makine modelini kullanmadan, hareketlerin anlık tanımlarla izlenmesidir.

Anlık Tanım (ID: Instantenaous Description)

Makinenin giriş dizgisi w olsun. Başlangıçta okuma kafası w'nin ilk (en soldaki) simgesi üzerindedir. Belirli sayıda hareket sonunda, okuma kafası w'nin belirli bir simgesinin üzerinde bulunur. Bu durumda, w'nin bir kesimi okuma kafasının solunda yer alır. w'nin okuma kafasının solunda kalan kesiminin α ile, geri kalan kesimini ise β ile gösterelim. Buna göre okuma kafası β'nın ilk (en soldaki) simgesi üzerinde bulunmaktadır.



Anlık tanım makinenin bulunduğu durum ile giriş dizgisinin okuma kafasının solunda ve sağında kalan kesimlerini gösteren bir üçlüdür.

$$ID = (\alpha, q, \beta)$$

Eğer ID1'den tek bir hareketle ID2'ye geçiliyorsa, bu iki anlık tanım arasındaki ilişki:

$$ID_1 \vdash ID_2$$

ile gösterilir.

Eğer ID1'den bir dizi hareket sonunda ID2'ye geçilebiliyorsa bu ilişki de:

$$ID_1 \vdash^* ID_2$$

ile gösterilir.

Anlık tanım ve anlık tanımlar arasındaki geçiş ilişkisine dayalı olarak iki yönlü sonlu bir özdevinirin (2DFA) tanıdığı dizgiler kümesi aşağıdaki gibi tanımlanır:

$$T(M) = \{ w \mid (q_0, w) \vdash^* (w, q_i), \ q_i \in F \}$$

Yukarıdaki tanımda, (q_0,w) başlangıç anlık tanımıdır (ID0). Başlangıçta okuma kafası w'nin ilk simgesi üzerinde bulunduğu ve okuma kafasının solunda dizginin hiçbir simgesi bulunmadığı için ID0'da α yoktur. (w,q1) ise, makinenin bir uç durumda ($q_1 \in F$) okuma kafasının ise w'nin sağındaki boş hücre üzerinde bulundupu son ID'dir. Son ID'de, w'nin tümü okuma kafasının solunda bulunduğu için bu kez β yoktur.

Anlık tanımları kullanarak, w_1=100101 ve w_2=100110 giriş dizgilerini iki yönlü sonlu özdevinir olan $M_{1.7}$ makinesi tarafından tanınıp tanınmadığını inceleyelim:

a) $w_1 = 1000101$

$(q_0, 1000101) \vdash (1, q_1, 000101) \vdash (10, q_1, 00101) \vdash (100, q_1, 0101) \vdash$
$(1000, q_1, 101) \vdash (100, q_2, 0101) \vdash (1000, q_0, 101) \vdash (10001, q_1, 01) \vdash$
$(100010, q_1, 1) \vdash (10001, q_2, 01) \vdash (100010, q_0, 1) \vdash (1000101, q_1)$

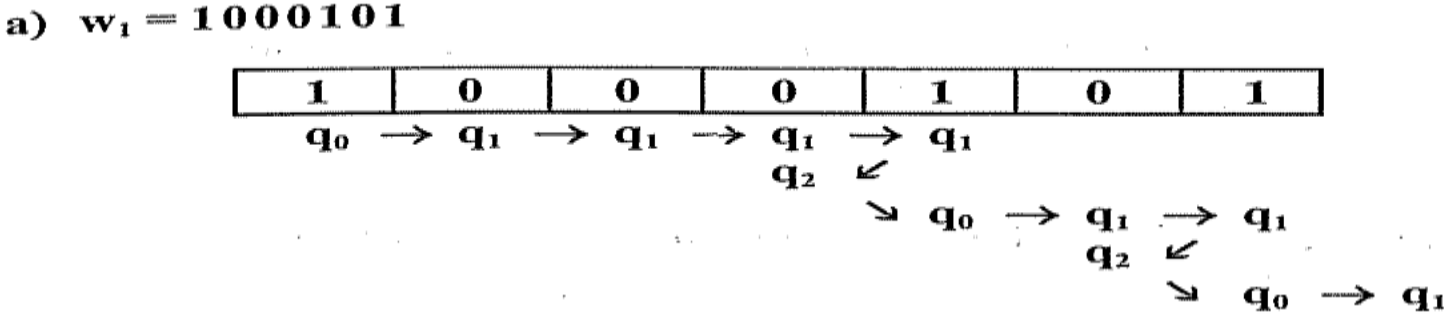
Sonuç : $w_1 = 1000101$ dizgisi $M_{1.7}$ tarafından tanınır.

b) $w_2 = 100110$

$(q_0, 100110) \vdash (1, q_1, 00110) \vdash (10, q_1, 0110) \vdash (100, q_1, 110) \vdash$
$(10, q_2, 0110) \vdash (100, q_0, 110) \vdash (1001, q_1, 10) \vdash (100, q_2, 110) \vdash$
$(10, q_2, 0110) \vdash (100, q_0, 110)$ .....

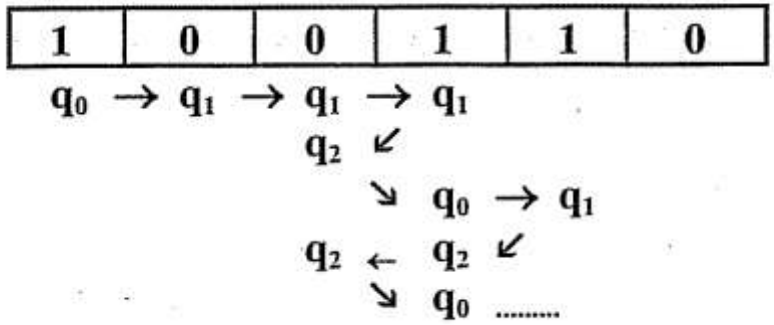
Sonuç : **ID** geçişlerinde döngü oluştuğu için $w_2 = 100110$ dizgisi $M_{1.7}$ tarafından tanınmaz.

Giriş dizgilerinin 2DFA tarafından tanınıp tanınmadığı ID'ler arası geçiş ilişkileri yerine, şerit üzerindeki hareketleri ve durum değişimlerini inceleyerek de bulabiliriz. Bu yöntemle w_1=100101 ve w_2=100110 giriş dizgilerinin $M_{1.7}$ makinesi tarafından tanınıp tanınmadığı aşağıdaki gibi bulunur:



Sonuç: w_1 = 100101 dizgisi $M_{1.7}$ tarafından tanınır.

b) $w_2 = 100110$



Sonuç: Makinenin hareketlerinde döngü oluştuğu ve w_2 = 100110 dizgisinin $M_{1.7}$ tarafından tanınmadığı yukarıda açık biçimde görülmektedir. Şeridin bir hücresi altındaki durumlar dizisinde aynı durumun birden çok kez yer alması döngünün göstergesidir.

Tek ve İki Yönlü Özdevinirlerin Denkliği

Daha önce, deterministik olmayan her sonlu özdevinire denk (aynı kümeyi tanıyan) deterministik bir sonlu özdevinirin bulunabileceği, dolayısıyla NFA ve DFA modellerin denk olduğunu gördük. Benzer biçimde, iki yönlü sonlu her özdevinire denk, tek yönlü bir sonlu özdevinir bulunabilir. Dolayısıyla 2DFA ve DFA modellerinin ifade güçleri denktir. İki modelin denkliği verilen bir 2DFA'ya denk tek yönlü bir DFA'nın bulunmasını sağlayan yöntem ile ispat edilebilir. Böyle bir yöntem vardır. Ancak bu yönteme bu kitapta yer verilmeyecektir

1.2.Çıkış Üreten Özdevinirler

Bölümün başında sonlu özdevinirlerin öncelikle:

a) Sonlu durumlu tanıyıcı (finite state recognizer)
b) Çıkış üreten özdevinir

modelleri olarak sınıflandırıldığı belirtilmişti. Tanıyıcı model, bir giriş alfabesindeki simgelerden üretilebilecek dizgilerin bir kesimini tanıyan bir modeldir. Bu model biçimsel dillerle ilgili söz dizim çözümleme (*syntax analysis*), ayrıştırma (*parsing*) ve benzeri uygulamalar başta olmak üzere birçok yazılım uygulamasında kullanılır. Sonlu durumlu tanıyıcı modeli deterministik ve deterministik olmayan türlerinin bulunduğu ve bu iki türün modelleme gücü açısından denk olduğu da yukarıdaki kesimlerde görüldü.

Çıkış üreten özdevinir modeli, adından da anlaşıldığı gibi girişine uygulanan bir giriş dizgisine yanıt olarak bir çıkış dizgisi üreten bir modeldir. Bu açıdan çıkış üreten özdevinir modelinin, giriş dizgilerini çıkış dizgilerine dönüştüren bir model olarak görmek de mümkündür. Böylece sonlu özdevinirlerin iki ana türü "tanıyıcılar" ve "dönüştürücüler" olarak nitelenebilir. Aslında tanıyıcıların da çıkış ürettikleri düşünülebilir. Mademki bir tanıyıcı girişine uygulanan giriş dizgilerinin bir kısmını tanımakta, diğer bir kısmını ise tanımamaktadır. FA modeli ikili çıkış üreten bir model olarak görülebilir. Modelin ürettiği çıkışlar örneğin uç durumlar için "1: tanıdı", diğer durumlar için "0:tanımadı" olabilir. Bu açıdan düşünüldüğünde ise çıkış üreten sonlu özdevinir modeli tanıyıcı modeli de kapsayan daha geniş bir model olarak görülebilir.

Çıkış üreten özdevinirleri Moore ve Mealy makinesi olarak adlandırılan iki türü vardır. Moore makinesi durum düzeyinde çıkış üreten bir model iken Mealy makinesi durum geçişi düzeyinde çıkış üreten bir modeldir. Aşağıda Moore ve Mealy makineleri ayrı ayrı incelenmektedir.

Moore Makinesi

- Moore makinesi bir altılı olarak tanımlanır:

  M=<Q, ∑, Δ, δ, λ, **q0>**

  Q: Sonlu durumlar kümesi

  ∑: Giriş alfabesi (sonlu bir küme)

  Δ: Çıkış alfabesi (sonlu bir küme)

  δ: Durum geçiş işlevi: (Q x ∑)'dan Q'ya bir eşleme

  λ: Çıkış işlevi: Q'dan Δ'ya bir eşleme

  q_0: Başlangıç durumu: q0 ⊆ Q

Moore makinesi DFA modelini genellemesi olarak görülebilir. Başka bir deyişle DFA modeli özel bir Moore makinesi olarak düşünülebilir. Gerçekten de eğer Moore modelinde çıkış alfabesi olarak {0,1} alfabesi alınır ve çıkış işlevi ile uç durumlara 1, uç olmayan durumlara da 0 değeri eşlenirse:

$$\Delta = \{0, 1\} \qquad \forall q_i \in F \Rightarrow \lambda(q_i) = 1 \qquad \forall q_i \notin F \Rightarrow \lambda(q_i) = 0$$

DFA modeli elde edilir.

Moore makinesinin durum geçiş işlevi (δ) ile çıkış işlevi (λ) bir çizelge ya da çizenek ile gösteriebilir. Moore makinesini tanımlamak için oluşturulan çizelge çoğunlukla "Durum Çizelgesi (*State Table*)" çizenek ise çoğunlukla "Durum Çizeneği (*State Diagram*)" olarak adlandırılır. Örnek 1.8'de tanımlanan $M_{1.8}$ Moore makinesini durum çizenek ve çizelgesi Çizim 1.9'da görülmektedir.

Örnek 1.8.

$M_{1.8}$ makinesi, girişine uygulanan ikili sayı X ise çıkışında z=Mod(X,5) değerinin üreten Moore makinesi olarak tanımlanıyor.

$$M_{1.8} = \langle Q, \sum, \Delta, \delta, \lambda, q0 \rangle$$

$$Q = \{A, B, C, D, E\}$$

$$\sum = \{0, 1\}$$

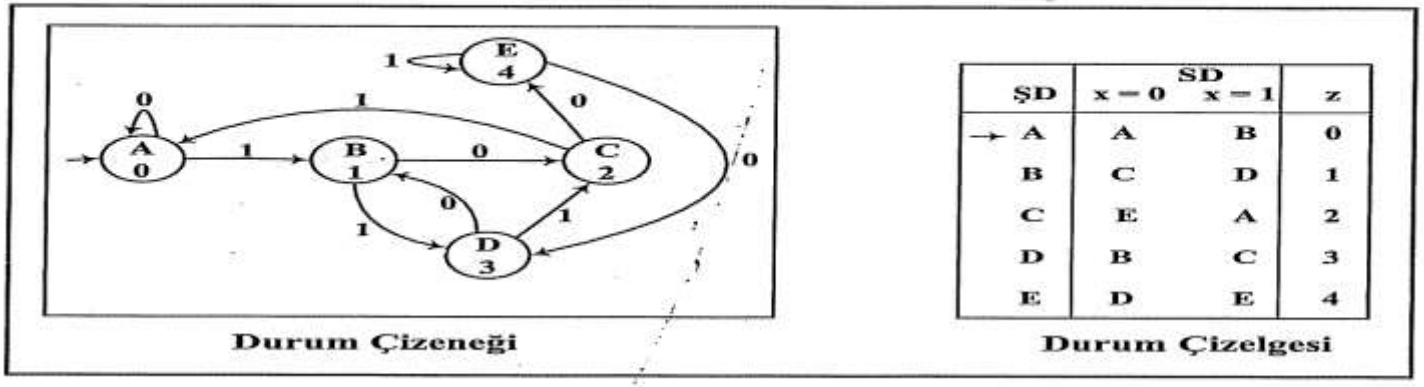
$$\Delta = \{0,1,2,3,4\}$$

$$q_0 = A$$

δ ve λ'nın tanımı, Çizim 1.9.a'daki durum çizeneği ile Çizim 1.9.b'deki durum çizelgesinde yer almaktadır.

$M_{1.8}$ makinesine örneğin w = 1011110 giriş dizgi uygulandığında makinenin durumları ve üreteceği çıkışlar aşağıdaki gibi değişecektir.

Giriş :		**1**		**0**		**1**		**1**		**1**		**1**		**0**
Durum:	**A**	→	**B**	→	**C**	→	**A**	→	**B**	→	**D**	→	**C**	→	**E**
Çıkış :	**0**		**1**		**2**		**0**		**1**		**3**		**2**		**4**



ŞD	SD x = 0	SD x = 1	z
→ A	A	B	0
B	C	D	1
C	E	A	2
D	B	C	3
E	D	E	4

Durum Çizelgesi

Çizim 1.9. Örnek Moore Makinesi ($M_{1.8}$)

Mealy Makinesi

Moore makinesi gibi Mealy makinesi de bir altılı olarak tanımlanır.

$$\mathbf{M = <Q, \sum, \Delta, \delta, \lambda, q0>}$$

Altılının çıkış işlevi (λ) dışındaki beş elemanının tanımı Moore makinesindekilerle aynıdır. Başka bir deyişle Moore ve Mealy makineleri arasındaki tek fark çıkış işlevindedir. Moore makineleri için Q'dan Δ'ya bir eşleme olarak tanımlanan çıkış işlevi Mealy makineleri için (Q x $\sum$)'dan Δ'ya bir eşleme olarak tanımlanır.

Örnek 1.9.

$M_{1.9}$ makinesi giriş alfabesi {0,1} çıkış alfabesi ise {0,1,2} olan ve ürettiği çıkış ile son iki giriş simgesinden kaç tanesinin değerini bir öncekinden farklı olduğunu gösteren Mealy makinesi olarak tanımlanıyor. Makinenin üreteceği ilk iki çıkış değeri belirlenirken başlangıç durumundan önceki iki giriş değerinin 00 olduğu varsayılacaktır.

$$\mathbf{M = <Q, \sum, \Delta, \delta, \lambda, q0>}$$

$$\mathbf{Q = \{A, B, C, D\}}$$

$$\sum = \mathbf{\{0, 1\}}$$

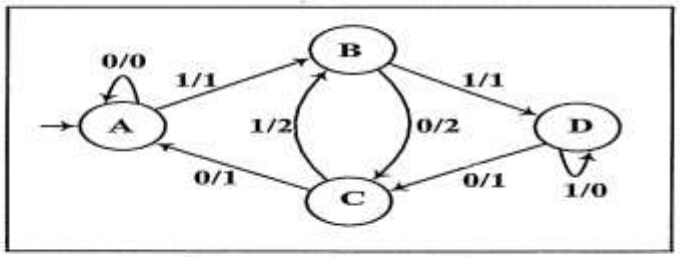
$$\Delta = \mathbf{\{ 0, 1, 2\}}$$

$$\mathbf{q0 = A}$$

δ ve λ, Çizim 1.10.a'daki Durum Çizeneği ve Çizim 1.10.b'deki Durum Çizelgesi ile tanımlanmaktadır. Yapılan tasarıma göre makinenin durumları önceki iki giriş simgesi değerini 00, 01, 10 ya da 11 olmasına göre sırasıyla A, B, C ya da D olmaktadır.

Örnek bir giriş dizgisiyle bu giriş dizgisine karşılık makinenin üretmesi gereken çıkış dizgisi aşağıdaki gibidir:

X: 0 1 1 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 0 1
Z: 0 1 1 1 2 2 2 2 2 1 0 0 1 2 2 1 1 1 0 1 1 0 0 1



Durum Çizeneği

ŞD	SD, z x = 0	x = 1
→ A	A, 0	B, 1
B	C, 2	D, 1
C	A, 1	B, 2
D	C, 1	D, 0

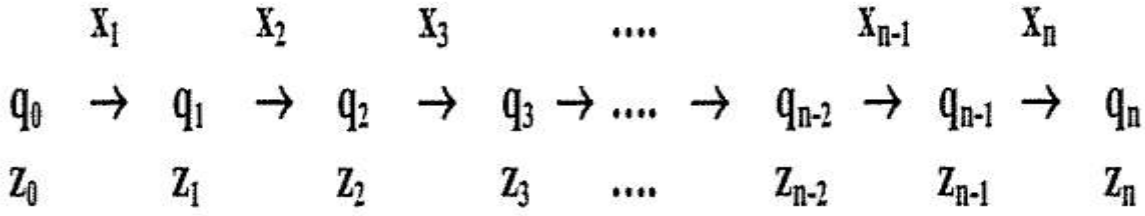
Durum Çizelgesi

Çizim 1.10. Örnek Mealy Makinesi ($M_{1.9}$)

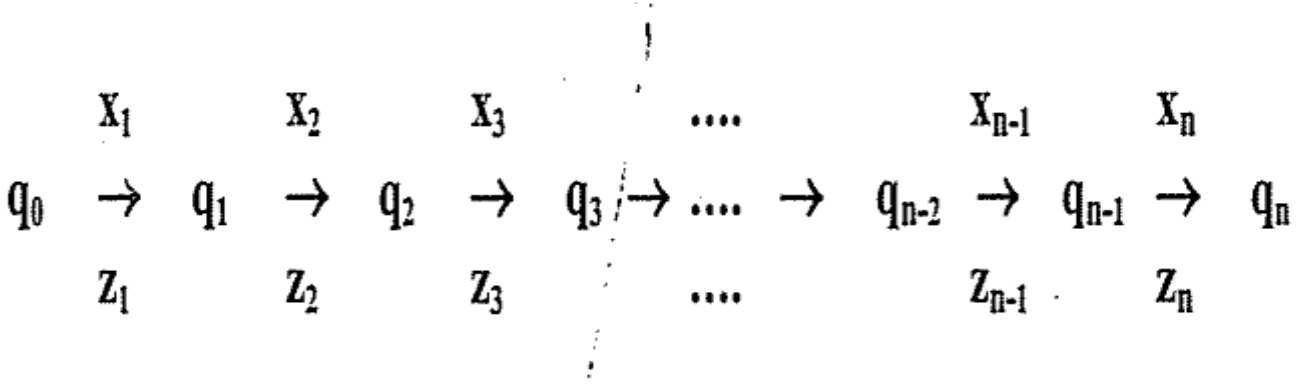

POWTOON

Moore ve Mealy Makinelerinin Eşdeğerliği

Moore ve Mealy makinelerinin her ikisi de çıkış üreten sonlu özdevinir modelidir. Birer altılı olarak tanımlanan Moore ve Mealy modellerinde altılının beş elemanı ortaktır. Moore ve Mealy modellerinin ayıran çıkış işlevidir. Moore modelinde, makine her durum için bir çıkış simgesi üretmektedir. Belirli bir durumda bulunan bir Moore makinesine, n giriş simgesinden oluşan bir dizgi uygulandığında, makine(n+1) uzunluğunda bir çıkış dizgisi üretecektir. Başlangıçtaki durumda q0, uygulanan n giriş simgesi sonrasındaki durumlar da q_1, q_2, …,q_n ile gösterilerek, Moore makinesinin giriş dizgisi, durum geçişleri ve çıkış dizgisi ilişkisi aşağıdaki gibi özetlenebilir. Hiçbir giriş simgesi uygulamadan, bulunduğu başlangıç durumuna göre Moore makinesinin bir çıkış simgesi (z_0) ürettiği dikkati çekmektedir.



Mealy modelinde ise makine her durum geçişi için bir çıkış simgesi üretmektedir. Belirli bir durumda bulunan bir Mealy makinesine, n giriş simgesinden oluşan bir giriş dizgisi uygulandığında, makine n uzunluğunda bir çıkış dizgisi üretecektir. Çünkü Mealy modelinde, giriş simgesi uygulanmadığı sürece makine çıkış üretmemektedir. Başlangıçtaki durum q0, uygulanan n giriş simgesi sonrasındaki durumlar da $q_1, q_2, \ldots, q_n$ ile gösterilerek, Mealy makinesinin giriş dizgisi, durum geçişleri ve çıkış dizgisi ilişkisi aşağıdaki gibi özetlenebilir.



Buna göre n uzunluğundaki X giriş dizgisine karşılık Mealy makinesi zmealy çıkış dizgisinin üretirken, Moore makinesi z_0 Z_{moore} çıkış dizgisini üretmektedir. Z_{mealy} ve Z_{moore} çıkış dizgiler n uzunluğunda dizgilerdir.

Eğer aynı giriş ve aynı çıkış alfabesine sahip bir Mealy ve bir Moore makinesinin ürettiği Z_{mealy} ve Z_{moore} çıkış dizgileri, tüm X giriş dizgileri için aynı ise bu iki makine birbirine denktir. Başka bir deyişle, Moore makinesinin başlangıçta ürettiği z_0 çıkış simgesi dışarda tutulursa uygulanan giriş dizgisi ne olursa olsun birbirine denk Mealy ve Moore makineleri aynı çıkış dizgisini üretecektir.

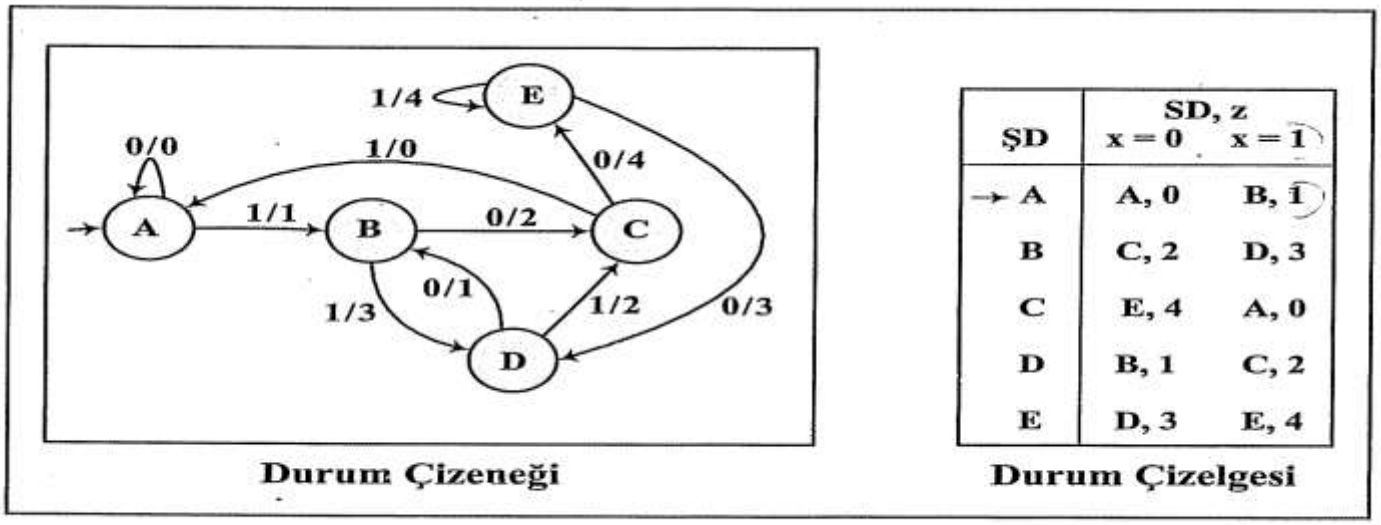
Moore Makinesine Eşdeğer Mealy Makinesinin Bulunması

Önerme 1.1. $M_1 = <Q, \sum, \Delta\ \delta, \lambda, q_0>$ bir Moore makinesi olsun. M_1'e eşdeğer Mealy makinesi aşağıdaki gibi bulunur.

$$M_2 = <Q, \sum, \Delta, \delta, \lambda, q_0>, \quad \lambda'(q, a) = \lambda(\delta(q, a))$$

Yukarıdaki tanıma göre bir Moore makinesi ile bu makineye eşdeğer Mealy makinesi arasındaki tek fark çıkış işlevindedir. Bilindiği gibi Moore makinesinde her duruma bir çıkış simgesi eşlenirken, Mealy makinesinde her durum geçişine (durum, giriş simgesi çiftine) bir çıkış simgesi eşlenir. Bu kitapta önermenin ispatına yer verilmeyecektir. Bir Moore makinesine eşdeğer Mealy makinesi bulunurken, her durum geçişine eşlenen çıkış değeri, durum geçişinin ucundaki sonraki duruma Moore makinesinde eşlenen çıkış değerine eşittir. Bu kuralı aşağıdaki çizimle özetlemek mümkündür.

Örnek 1.8'deki Moore makinesine ($M_{1.8}$) eşdeğer Mealy makinesinin, Önerme 1.1'e göre bulunan durum çizenek ve çizelgesi Çizim 1.11'de görülmektedir. bir Moore makinesi ile bu makineye eşdeğer Mealy makinesinin hem durum sayılarının hem de durumlar arası geçiş sayılarının eşit olduğu görülmektedir.



ŞD	SD, z x = 0	x = 1
→ A	A, 0	B, 1
B	C, 2	D, 3
C	E, 4	A, 0
D	B, 1	C, 2
E	D, 3	E, 4

Durum Çizelgesi

Çizim 1.11. $M_{1.8}$ Moore Makinesine Eşdeğer Mealy Makinesi

Mealy Makinesine Eşdeğer Moore Makinesinin Bulunması

Önerme 1.2. $M_2 <Q, \sum, \Delta\ \delta, \lambda, q_0>$ bir Mealy makinesi olsun. M_2'ye eşdeğer M_1 Moore makinesi aşağıdaki gibi bulunur:

$M_1 = <Q', \sum', \Delta', \delta', \lambda', q_0'>$

$Q' = Q \times \Delta$

$q_0' = [q_0, zj]$ (zj = çıkış simgelerinden rastgele seçilmiş biri)

$\delta'([q_1, z_k], x_j) = [\delta(q_1, x_j)], \lambda(q_1, x_j)]$

$\lambda'([q_1, z_k]) = z_k$

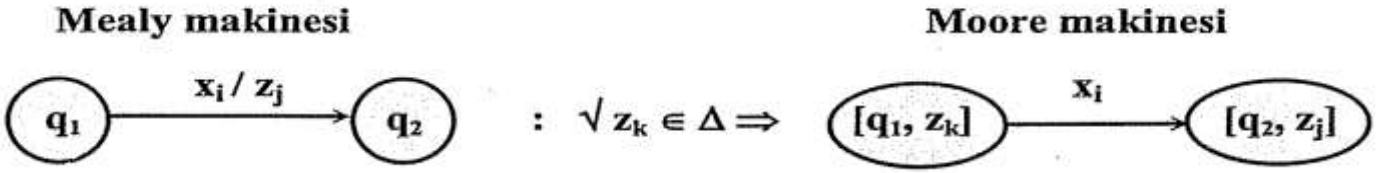
Önerme 1.2'ye göre bir Mealy makinesi (M_2) ile bu makineye eşdeğer Moore makinesinin (M_1) tanımını oluşturan altılıların dördü birbirinden farklıdır.

- M_2'nin her [durum çıkış simgesi] çiftine karşılık M_1 makinesinin bir durumu vardır. Buna göre M_2 makinesinin durum sayısı n, çıkış simgesi sayısı ise m ise M_1 makinesinin n x m durumu bulanacaktır. Örneğin M_9 Mealy makinesinin 4 durumu, 4 de çıkış simgesi vardır. M_9 Mealy makinesinin, Önerme 1.2'ye göre bulunacak eşdeğeri Moore makinesinin 4x4 = 16 durumu olacak ve bu durumlar sembolik olarak [A,0], [A,1], …, [D,4] diye gösterilmektedir.
- M_1'in durumlarından, sembolik gösterimde birinci elemanı q_0 olanlar arasından rastgele seçilen biri başlangıç durumu yapılır. Örneğin çıkış alfabesinde 4 simge bulunan M_9 Mealy makinesinin başlangıç durumu A olduğuna göre bu makineye eşdeğer Moore makinesinin başlangıç durumu [A,0], [A,1], [A,2] ve [A,3] arasından rastgele seçilen biri olacaktır.
- M_1'in geçiş işlevi

$$\delta' ([q_1, z_k], x_j) = [\delta(q_1, x_j)], \lambda(q_1, x_j)]$$

olarak tanımlanmaktadır.

Bu tanımın daha kolay anlaşılmasını sağlamak için aşağıdaki çizim oluşturulabilir.



Bir Mealy makinesine eşdeğer Moore makinesinin, yukarıdaki çizimle açıklamaya çalışılan durum geçiş işlevi, sözlü olarak da şöyle anlatılabilir. Mealy makinesinde q_1 durumundan q_2 durumuna z_j geçişi sırasında z_j çıkış simgesi üretiliyorsa eşdeğer Moore makinesinde birinci elemanı q_1 olan her durumdan x_j giriş simgesi ile birinci elemanı q_2, ikinci elemanı ise z_j olan duruma bir geçiş oluşturulur. Buna göre eğer çıkış alfabesinde m çıkış simgesi varsa, Mealy makinesindeki her durum geçişine karşılık, eşdeğer Moore makinesinde m geçiş bulunacaktır. Böylece bir Mealy makinesinin her durumuna karşılık eşdeğer Moore makinesinde m durum bulunduğu gibi, Mealy makinesindeki her durum geçişine karşılık eşdeğer Moore makinesinde m durum geçişi bulunacaktır.

Örnek 1.10 Mealy makinesi aşağıdaki gibi tanımlanıyor:

$M_{1.10} = <Q, \sum, \Delta, \delta, \lambda, q_0>$

$Q = \{A, B, C\}$

$\sum = \{0, 1\}$

$\Delta = \{0, 1\}$

$q_0 = A$

$\delta(A, 0) = B$	$\lambda(A, 0) = 0$
$\delta(A, 1) = A$	$\lambda(A, 1) = 1$
$\delta(B, 0) = B$	$\lambda(B, 0) = 1$
$\delta(B, 1) = C$	$\lambda(B, 1) = 1$
$\delta(C, 0) = A$	$\lambda(C, 0) = 0$
$\delta(C, 1) = C$	$\lambda(C, 1) = 0$

$M_{1.10}$ Mealy makinesine eşdeğer Moore makinesini ($M'_{1.10}$) bulalım.

$M'_{1.10} = < Q', \sum, \Delta, \delta', \lambda', q'_0>$

$Q' = \{[A, 0], [A, 1], [B, 0], [B, 1], [C, 0], [C, 1]\}$

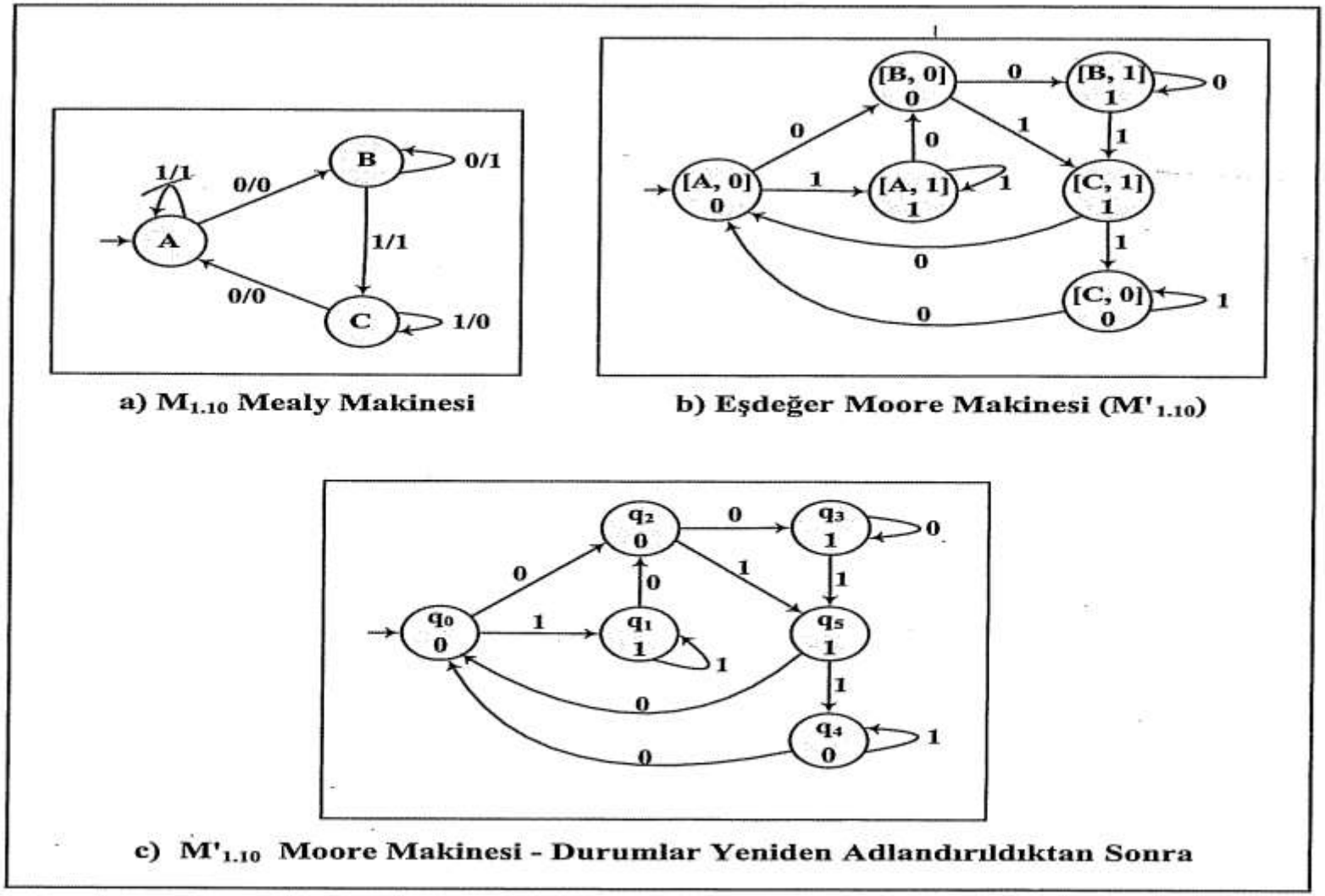
$q'_0 = [A, 0]$

$\delta'([A, 0], 0) = [B, 0]$	$\lambda[A, 0] = 0$
$\delta'([A, 1], 0) = [B, 0]$	$\lambda[A, 1] = 1$
$\delta'([A, 0], 1) = [A, 1]$	$\lambda[B, 0] = 0$
$\delta'([A, 1], 1) = [A, 1]$	$\lambda[B, 1] = 1$
$\delta'([B, 0], 0) = [B, 1]$	$\lambda[C, 0] = 0$
$\delta'([B, 1], 0) = [B, 1]$	$\lambda[C, 1] = 1$
$\delta'([B, 0], 1) = [C, 1]$	
$\delta'([B, 1], 1) = [C, 1]$	
$\delta'([C, 0], 0) = [A, 0]$	
$\delta'([C, 1], 0) = [A, 0]$	
$\delta'([C, 0], 1) = [C, 0]$	
$\delta'([C, 1], 1) = [C, 0]$	

Çizim 1.12'de $M_{1.10}$ Mealy makinesi ile bu makineye eşdeğer Moore makinesinin durum çizeneği görülmektedir. Önerme 1.2'ye göre bir Mealy makinesine eşdeğer Moore makinesini bulurken, Moore makinesinin durumlarının [durum adı, çıkış simgesi] ikilileri ile gösterilmesi gerekir. Ancak eşdeğer Moore makinesi bulunduktan sonra, bu makinenin durumları yeniden adlandırılabilir. Nitekim, $M_{1.10}$ Moore makinesinin Çizim 1.12.c'e görülen durum çizeneği, makinenin durumları:

[A,0] için q_0
[A,1] için q_1
[B,0] için q_2
[B,1] için q_3
[C,0] için q_4
[C,1] için q_5

adları kullanılarak oluşturulmuştur.



Çizim 1.12. Mealy Makinesine Eşdeğer Moore Makinesinin Bulunması


POWTOON

1.3.Sonlu Özdevinirlerin İndirgenmesi

Sonlu özdevinirlerle ilgili olarak şimdiye kadar birden çok model tanımlandı. Tanımlanan modelleri aşağıdaki gibi özetlemek mümkündür:

Tanıyıcı model. Sonlu özdevinirlerin temel modeli olan ve kısaca FA olarak adlandırılan bu modelin deterministik (DFA) ve deterministik olmayan (NFA) türleri vardır.

Çıkış üreten özdevinir modeli. Bu modelin de Mealy ve Moore modeli (ya da makinesi) olarak adlandırılan iki türü bulunmaktadır.

Gerek FA modelinde gerekse çıkış üreten sonlu özdevinir modelinde tanımlanan sonlu özdevinirin (makinenin) karmaşıklığı durum sayısı ile doğru orantılıdır. Bu nedenle, bir sonlu özdevinir verildiğinde bu sonlu özdevinirin indirgenmesi önem taşıyabilir. Bir sonlu özdevinirin indirgenmesi ya da yalınlaştırılması, bu sonlu özdevinire denk, durum sayısı en küçük sonlu özdevinirin bulunmasıdır. Sonlu özdevinirlerin indirgenmesi DFA, Moore ve Mealy modelleri için görülecektir. Bunun için önce belirli tanımların yapılması gerekir.

1.3.1. Ardıl, Öncel, Denk ve Ayırt Edilebilir Durum Tanımları

DFA, Moore ve Mealy modellerinin en genişi Mealy modelidir. Çünkü tanıyıcı (FA) model, Moore modelinin çıkış alfabesi {kabul, red} olan bir alt türü olarak görülebilir. Moore modeli ise, Mealy modelinin, çıkış işlevi giriş alfabesinden bağımsız olan bir alt türü olarak görülebilir. Bu nedenle tanımlar verilirken Mealy modeli esas alınacaktır ve örnek olarak aşağıda tanımı verilen $M_{1.11}$ makinesi kullanılacaktır.

Örnek 1.11

Mealy türü $M_{1.11}$ makinesinin giriş ve çıkış alfabeleri {0, 1} alfabesidir. Başlangıç durumu A olan makinenin geçiş ve çıkış işlevleri aşağıda durum çizelgesi ile tanımlanmaktadır.

ŞD	SD, z	
	x = 0	x = 1
➔ A	A, 0	D, 1
B	C, 0	E, 1
C	G, 0	E, 1
D	G, 0	F, 1
E	E, 1	C, 0
F	B, 0	D, 1
G	B, 0	E, 1

Ardıl

M makinesinin S_1 durumundan x giriş simgesi ile S_2 durumuna geçiliyorsa:

S_1'in x-ardılı S_2'dir. w bir giriş simgeleri dizisi olmak üzere, eğer S_1 durumundan w giriş dizgisi ile S_2 durumuna geçiliyorsa, S_1'in w-ardılı S_2'dir. Örneğim $M_{1.11}$ makinesinde A durumunun 1-ardılı D'dir. Aynı makine B durumunun 011-ardılı ise C'dir.

X bir giriş simgesi, w ise giriş simgelerinin bir dizgisi olmak üzere, deterministik modellerde, bir durumun x ve w ardılı her zaman tek bir durumdur. Deterministik olmayan modellerde ise x ve w ardılları, durumların birer altkümesidir. Geçiş çizeneğinde yer aldığı için bir durumun x-ardılı kolaylıkla bulunabilir. Bir durumun $w=x_1x_2\ldots x_k$ ardılını bulmak için ise, geçiş çizeneği kullanılarak önce x1-ardılı bulunur; daha sonra x_1-ardılın x_2-ardılı, … vb bulunarak w-ardılı elde edilir. Örneğin $M_{1.3}$ makinesinde q_0 durumunun 0-ardılı $\{q_0, q_1\}$'dir. $\{q_0,q_1\}$ yerine q_0q_1 yazılabilir. Buna göre q_0 durumunun 0-ardılı q_0q_1, 1-ardılı q_0q_2, 100-ardılı ise $q_0q_1q_3$'dür. Aynı makinenin q_1 durumunun 1-ardılı ile q_2 durumunun 01-ardılı ise yoktur. (boş kümedir.)

Öncel

M makinesinin S_1,S_2, … ve S_i durumlarından x giriş simgesi ile S_j durumuna geçiliyorsa, S_j durumunun x-önceli, {S_1,S_2,…, S_j}'dir. Kullanılan model deterministik olsa bile, bir durumun x ve w öncelleri, durumların birer altkümesidir. Örneğin $M_{1.11}$ makinesi için B durumunun 0-önceli FG, 001-önceli de AEF'dir. Aynı makine için D durumunun 0 ve 110 öncelleri ise yoktur. Bir makinenin x-öncellerini bulmak için öncel çizelgesi adı verilen bir çizelge oluşturulabilir. $M_{1.11}$ makinesi için oluşturulan öncel çizelgesi aşağıda görülmektedir.

ŞD	Önceki Durum	
	x = 0	x = 1
A	A	-
B	FG	-
C	B	E
D	-	AF
E	E	BCG
F	-	D
G	CD	-

Denk Durumlar

M makinesi S_1 ve S_2 durumlarından herhangi birinde iken, hangi giriş simgesi uygulanırsa uygulansın makine hep aynı çıkış simgesi üretiyorsa, bu durumlara *1-denk durumlar* denir. Örneğin $M_{1.11}$ makinesinin A ve B durumları 1-denktir. C ve F durumları da 1-denktir. Buna karşılık E ve G durumları 1-denk değildir.

M makinesi S_1 ve S_2 durumlarından herhangi birinde iken, uzunluğu n ya da daha kısa olan hangi giriş dizgisi uygulanırsa uygulansın, makine hep aynı çıkış dizgisini üretiyorsa, bu durumlara *n-denk durumlar* denir. Örneğin $M_{1.11}$ makinesinin A ve D durumları 2-denktir. B ve C durumları 3-denktir. B ve C durumları aynı zamanda 4-denktir, 5-denktir,…vb. Buna karşılık A ve D durumları 3-denk değildir.

M makinesi S_1 ve S_2 durumlarından herhangi birinde iken, uzunluğuna bakılmaksızın, hangi giriş dizgisi uygulanırsa uygulansın, makine hep aynı çıkış dizgisini üretiyorsa, bu durumlara *denk durumlar* denir. Örneğin $M_{1.11}$ makinesinin B ve C durumları denk durumlardır. D ve F durumları da denktir. Ancak 2-denk olan A ve D durumları denk değildir.

Uygulamada, tüm n'ler için n-denk olan durumlara *denk durumlar* denir. N ve n'den küçük tüm k değerleri (√k<=n) için k-denk olan iki duruma ise *n-denk* denir. Örneğin $M_{1.11}$ makinesinin A ve F durumları 2-denk durumlardır. Bu durumların 2-denk olarak nitelenmesi, 3-denk olmadıkları anlamına gelir. Ancak 2-denk olan durumlar doğal olarak 1-denktir. Ancak durum denkliği her zaman en büyük n değeri üzerinde belirtilir.

Ayırt Edilebilir Durumlar

M makinesinin S_1 ve S_2 durumlarını ayırt etme için eğer en az n uzunluğunda bir giriş dizgisi gerekli ise bu durumlara *n-ayırt edilebilir durumlar* denir. Eğer S_1 ve S_2 durumları n-ayırt edilebilir ise, bu iki durum (n-1)-denktir. Örneğin $M_{1.11}$ makinesinin A ve E durumları 1-ayırt edilebilir. B ve D durumları ise 2-ayırt edilebilir. Buna karşılık A ve F durumları 3-ayırt edilebilir. D ve F durumlarını ise hiçbir giriş dizgisi ile ayırt etmek mümkün değildir. Denk olan D ve F durumları ayırt edilemez durumlardır.

$$A \xrightarrow[Z=0]{X=0} A \qquad E \xrightarrow[Z=1]{X=0} E \qquad \textbf{A ve E 1-ayırdedilebilir}$$

$$B \xrightarrow[Z=11]{X=10} E \qquad D \xrightarrow[Z=10]{X=10} B \qquad \textbf{B ve D 2-ayırdedilebilir}$$

$$A \xrightarrow[Z=010]{X=010} G \qquad F \xrightarrow[Z=011]{X=010} E \qquad \textbf{A ve F 3-ayırdedilebilir}$$

Makine Denkliği

M_1 ve M_2, giriş ve çıkış alfabeleri aynı olan iki makine olsun. M_1 makinesinin durumlarını S_{11}, S_{12}, S_{13},… olarak; M_2 makinesinin durumları ise S_{21},S_{22},S_{23},… olarak gösterelim. eğer M_1 makinesi S_{11}, M_2 makinesi de S_{21} durumundayken, her iki makineye aynı giriş dizgisi uygulandığında, giriş dizgisi ne olursa olsun her iki makine de aynı çıkış dizgisini üretiyorsa, bu ki durum (M_1 makinesinin S_{11} durumu ile M_2 makinesi de S_{21} durumu) denktir.

Eğer M_1 makinesinin her durumu için M_2 makinesinde bu duruma denk bir durum varsa; M_2 makinesinin her durumu için de M_1 makinesinde bu duruma denk bir durum varsa, M_1 ve M_2 makineleri denktir.

$$\left.\begin{array}{l} \sqrt{} S_{1i} \in M_1 \rightarrow \exists\, S_{2j} \in M_2 : S_{1i} \equiv S_{2j} \\ \sqrt{} S_{2i} \in M_2 \rightarrow \exists\, S_{1j} \in M_1 : S_{2i} \equiv S_{1j} \end{array}\right\} M_1 \equiv M_2$$

Makine İndirgeme

Bir M makinesi verildiğinde, bu makinenin indirgenmesi ya da yalınlaştırılması, bu makineye denk makinelerden durum sayısı en küçük olanının bulunması anlamı taşır. Bu makineyi M_{min} ya da M* ile gösterebiliriz. Bir makineye denk bir ya da birden çok en küçük makine olabilir.

1.3.2. İndirgeme Yöntemi

Sonlu özdevinirlerin indirgenmesi için denklik bölümleri (*equivalence partitions*) kullanılır. Bir M makinesi için, P_k ile gösterilen k-denklik bölümlemesi, k-denk durumların aynı bölümde yer aldığı bir bölümlemedir. Örneğin Mealy makinesi olan $M_{1.11}$ için 0-denklik bölümlemesi

$P_0 = (ABCDEFG)$

tek bölüm içerir. Çünkü hiçbir giriş simgesi uygulanmadan, bir Mealy makinesinin durumlarını ayırt etmek mümkün değildir. M1.11 için 1-denklik bölümlemesi ise

$P_1 = (ABCDFG)(E)$

iki bölüm içerir. Çünkü makinenin E dışındaki tüm durumları 1-denktir. Makinenin denklik bölümlemesini bulmak için sırasıyla P_0, P_1, P_2,.. bulunur. Denklik bölümlemelerinin türetilmesi

$P_{k+1} = P_k$

elde edilinceye kadar sürdürülür. $P_{k+1} = P_k$ elde edildiğinde, denklik bölümlenmesinin

$P = P_k$

olduğu anlaşılır.

Pm denklik bölümlemesinden, Pm+1 denklik bölümlemesinin türetilmesi için Önerme1.3'den yararlanılır.

Önerme1.3.

M makinesinin S_1 ve S_2 durumunun (m+1) denk olması için aşağıdaki iki koşulum sağlanması gerekli ve yeterlidir.

a. S_1 ve S_2 m-denk olmalı (P_m denklik bölümlemesinde aynı bölümde bulunmalı.)
b. Tüm x giriş simgeleri için S_1 ve S_2 durumlarının x-ardılları da m*denk olmalı (Pm denklik bölümlemesinde aynı bölümde bulunmalı.)

1.3.2.1. Mealy Makinelerinin İndirgenmesi

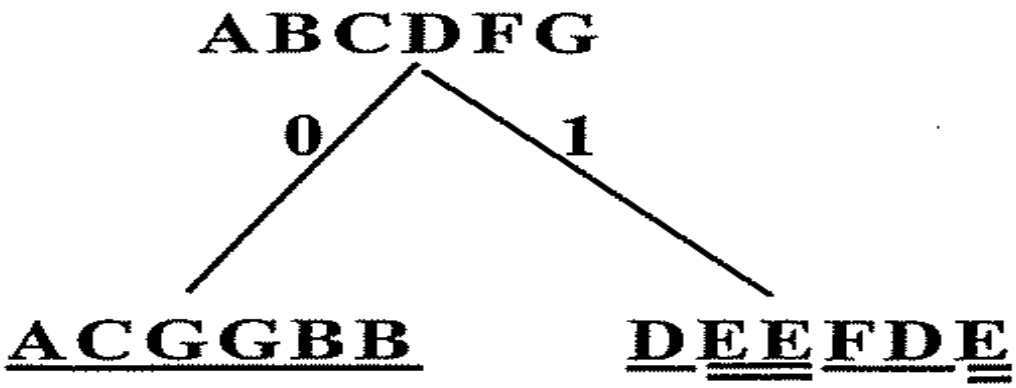
Mealy makinelerinin indirgenmesi M1.11 makinesi örnek alınarak incelenecektir. Daha önce belirtildiği gibi, hiçbir giriş uygulanmadan Mealy makinesinin durumları ayırt edilemez. Başka bir deyişle Mealy makinesinin tüm durumları 0-denktir.

$P_0 = (ABCDEFG)$

P_1 denklik bölümlemesini bulmak için durum çizelgesinin incelenmesi yeterlidir. M1.11'in durum çizelgesinde, E dışındaki tüm durumlar için, x geçişi sırasında üretilen çıkış değerinin birinci kolonda 0, ikinci kolonda ise 1 olduğu görülür. Buna göre E dışındaki durumlar 1-denktir.

$P_1 = (ABCDFG)(E)$

P_2 denklik bölümlemesi bulmak için Önerme 1.3'den yararlanılır. $M_{1.11}$ makinesinin iki durumunun 2-denk olabilmesi için, hem bu iki durumun, hem de bu iki durumun 0 ve 1-ardıllarının 1-denk olması; bunun için de P_1 denklik bölümlemesinde aynı bölümde bulunması gerekir.



P_1 bölümlemesinde aynı bölümde bulunan ABCDFG durumlarının 0-ardılları, P_1 bölümlemesinde aynı bölümde yer aldığı için 1-denktir. Ancak bu durumlarının 1- ardıllarının tümü 1-denk değildir. Buradan, ADF durumlarının kendi aralarında, BCG durumlarının da kendi aralarında 2-denk olduğu anlaşılır ve aşağıdaki P_2 denklik bölümlemesi elde edilir.

$P_2 = (ADF)(BCG)(E)$

P_3 denklik bölümlemesini bulmak için ADF ve BCG durumlarının 0 ve 1-adrılları incelenir:

Yapılan incelemede ADF durumlarının 1-ardıllarının P_2'de aynı bölümde yer aldığı, ancak 0-ardıllarının P_2'de aynı bölümde yer almadığı görülür. Bu nedenle DF durumlarının 3-denk olduğu, ancak A durumunun D ve F durumlarına 3-denk olmadığı anlaşılır ve P_3'de A durumu DF durumlarından ayırır. BCG durumlarına gelince, bu durumların hem 0 hem de 1-ardılları P_2'de aynı bölümde yer almaktadır. Buna göre BCG durumları 3-denk durumlardır ve P_3 denklik bölümlemesinde aynı bölümde yer alacaklardır. Sonuç olarak P_3 denklik bölümlemesi aşağıdaki gibi oluşur:

$P_3 = (A)(DF)(BCG)(E)$

P_4 denklik bölümlemesini bulmak için DF ve BCG durumlarının 0 ve 1-ardıllarını incelemek gerekir. BCG durumlarının, daha önce incelenen ve yukarıda yer alan 0 ve 1-ardılları P_3'de de aynı bölümde yer almaktadır. Bu nedenle BCG durumları P_4'de de aynı bölümde yer alacaktır.

DF durumlarının ardılları incelendiğince ise, bu iki durumun 0 ve 1-ardıllarının P_3'de aynı bölümde yer aldığı görülür. Bu nedenle DF durumları P_4'de aynı bölümde yer alacaktır. Sonuç olarak, P_4 denklik bölümlemesi P_3'e eşittir.

$P_4 = P_3 = (A)(DF)(BCG)(E)$

Böylece M1.11 makinesinin denklik bölümlemesi:

$P = (A)(DF)(BCG)(E)$

olarak elde edilir. Denklik bölümlemesinde 4 bölüm olduğu için, $M_{1.11'e}$ denk en küçük makinenin 4 durumu olacaktır. En küçük makinenin durumları:

A için S_0,

DF için S_1,

BCG için S_2,

E için S_3

Diye adlandırılırsa, $M_{1.11}$ makinesine denk en küçük makinenin durum çizelgesi aşağıdaki gibi bulunur.

ŞD	SD, z	
	x = 0	x = 1
S_0	S_0, 0	S_1, 1
S_1	S_2, 0	S_1, 1
S_2	S_2, 0	S_3, 1
S_3	S_3, 1	S_2, 0

1.3.2.2. Moore Makinelerinin İndirgenmesi

Bilindiği gibi Moore makinelerinde çıkış işlevi durumlar kümesinden çıkış alfabesine bir eşlemedir. Bu eşlemeyle her duruma bir çıkış simgesi eşlenir. Bir Moore makinesi belirli bir durumda iken belirli bir çıkış simgesi üretir. Bu nedenle hiçbir giriş simgesi uygulanmadan belirli durumlar ya da durum altkümeleri ayırt edilebilir. Başka bir deyişle, Moore makinesinin tüm durumları 0-denk değildir. Moore makinesinin P_0 0-denklik bölümlemesinde, çıkış simgesi kadar bölüm bulunur.

Mealy ve Moore makinelerinin indirgenmesindeki tek fark P_0 denklik bölümlemesinin oluşturulmasıdır. İndirgenmenin sonraki adımları benzer biçimde yürütülür. Moore makinelerinin indirgenmesi, örnek bir makine üzerinde görülecektir.

Örnek 1.12.

Moore türü $M_{1.12}$ makinesinin giriş alfabesi {0,1}, çıkış alfabesi ise {0,1,2} alfabesidir. Başlangıç durumu A olan makinenin geçiş ve çıkış işlevleri aşağıdaki durum çizelgesi ile tanımlanmaktadır.

ŞD	SD x = 0	SD x = 1	z
→ A	C	B	0
B	B	D	1
C	A	H	2
D	E	G	1
E	C	D	2
F	C	H	2
G	G	H	1
H	F	B	1

$M_{1.12}$ makinesi aşağıdaki gibi indirgenir:

$P_0 = (A)(BDGH)(CEF)$

$P_1 = (A)(BG)(DH)(C)(EF)$

$P_2 = P_1 = (A)(BG)(DH)(C)(EF)$

$P = (A)(BG)(DH)(C)(EF)$

Denklik bölümlemesinde 5 bölüm olduğu için, M1.12'ye denk en küçük makinenin 5 durumu olacaktır. En küçük makinen durumları:

A için S_0,

BG için S_1,

DH için S_2,

C için S_3,

EF için S_4

diye adlandırılırsa, $M_{1.12}$ makinesine denk en küçük makinenin durum çizelgesi aşağıdaki gibi bulunur.

ŞD	SD, z		
	x = 0	x = 1	z
→ S_0	S_3	S_1	0
S_1	S_1	S_2	1
S_2	S_4	S_1	1
S_3	S_0	S_2	2
S_4	S_3	S_2	2

1.3.2.3.Deterministik Sonlu Özdevinirlerin (DFA) İndirgenmesi

Bilindiği gibi DFA modeli, Moore makinesinin kısıtlı bir türüdür. Moore modelinde çıkış alfabesi {kabul, red} gibi ikili bir alfabeyle sınırlanırsa, durumlar çıkış işlevi ile “kabul eden” ve “kabul etmeyen” durumlar olmak üzere ikiye ayrılır ve DFA modeli elde edilir. Buna göre DFA’ların indirgenmesi Moore makinelerinin indirgenmesi ile aynı olacaktır.

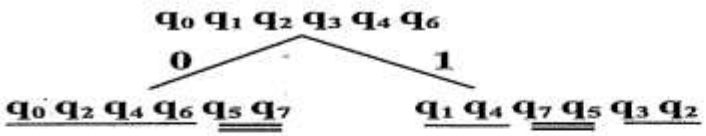
Önerme 1.13.

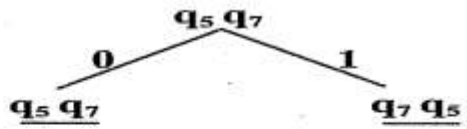
Deterministik bir özdevinir (DFA) olan M1.13 aşağıdaki durum çizelgesi ile tanımlanmaktadır.

ŞD	SD x = 0	x = 1
→q_0	q_0	q_1
q_1	q_2	q_4
q_2	q_4	q_7
q_3	q_6	q_5
q_4	q_5	q_3
(q_5)	q_5	q_7
q_6	q_7	q_2
(q_7)	q_7	q_5

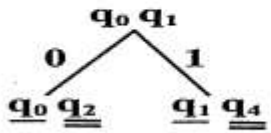
$M_{1.13}$ makinesi aşağıdaki gibi indirgenir:

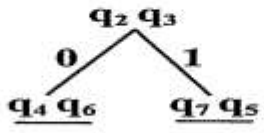
$P_0 = (q_0\ q_1\ q_2\ q_3\ q_4\ q_6)(q_5\ q_7)$

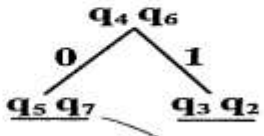


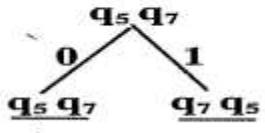


$P_1 = (q_0\ q_1)(q_2\ q_3)(q_4\ q_6)(q_5\ q_7)$









$P_2 = (q_0)(q_1)(q_2\ q_3)(q_4\ q_6)(q_5\ q_7)$

$P_3 = P_2$

$P = (q_0)(q_1)(q_2\ q_3)(q_4\ q_6)(q_5\ q_7)$

Denklik bölümlemesinde 5 bölüm olduğu için, $M_{1.13}$ denk en küçük makinenin 5 durumu olacaktır. En küçük makinenin durumları:

q_0 için S_0,

q_1 için S_1,

q_2q_3 için S_2,

q_4q_6 için S_3,

q_5q_7 için S_4

diye adlandırılırsa $M_{1.13}$ makinesine denk en küçük makinenin durum çizelgesi aşağıdaki gibi bulunur.

ŞD	SD, z	
	x = 0	x = 1
➔ S_0	S_0	S_1
S_1	S_2	S_3
S_2	S_3	S_4
S_3	S_4	S_2
S_4	S_4	S_4

BÖLÜM 2
DÜZGÜN KÜMELER ve DÜZGÜN DEYİMLER

2.1.Düzgün Kümeler

Sonlu özdevinir (*finite automata* : FA) modeli, giriş alfabesindeki simgelerden oluşan dizgilerin bir kısmını tanıyan, bir kısmını ise tanımayan bir modeldir. Her sonlu özdevinirin tanıdığı bir dizgiler kümesi vardır. Bu küme, içerdiği dizgiler tek tek yazılarak ya da içerdiği dizgilerin özellikleri belirtilerek tanımlanabilir. Sonlu özdevinirlerin tanıdığı kümelerin tanımları sözel olarak ya da matematiksel gösterimler kullanılarak yapılabilir. Örneğin {0, 1} alfabesinde, sonlu bir özdevinir tarafından tanınan aşağıdaki kümeleri tanımlayabiliriz:

$\mathbf{P_1}$ = {1001, 0110}

$\mathbf{P_2}$ = İçindeki art arda en az üç tane 1 bulunan (ya da 111 alt dizgisini içeren) dizgiler kümesi

$\mathbf{P_3}$ = İlk ve son simgesi 1 olan, en çok 5 uzunluğundaki dizgiler kümesi

$\mathbf{P_4} = \{\mathbf{0^n\ 1^m} \mid n>=1, m>=1\}$

Sonlu özdevinirler tarafından tanınan kümelere *düzgün kümeler (regular sets)* denir. Bir küme verildiğinde, bu kümenin düzgün bir küme olup olmadığı, bu kümeyi tanıyan sonlu bir özdevinirin bulunup bulunmaması tarafından belirlenir. Eğer kümeyi tanıyan en az bir sonlu özdevinir varsa, bu küme düzgün bir kümedir. Eğer verilen kümeyi tanıyan hiçbir sonlu özdevinir yoksa, bu küme düzgün bir küme değildir. Yukarıda tanımlanan $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ ve $\mathbf{P_4}$ kümeleri düzgün kümelerdir. Çünkü bu kümelerin her birini tanıyan en az bir sonlu özdevinir oluşturulabilir.

$\mathbf{P_5} = \{\mathbf{0^n\ 1^m} \mid n>=1\}$

$\mathbf{P_6}$ = İçinde eşit sayıda 0 ve 1 bulunan dizgiler kümesi

P_5 ve P_6 kümelerinin her ikisi de düzgün değildir. Çünkü bu kümeleri tanıyan sonlu özdevinirler oluşturulamaz. Tanımlanan bir kümenin düzgün olup olmadığını bulmak için, kümeyi tanıyan bir sonlu özdevinir oluşturulmaya çalışılır. Eğer bunda başarılı olunur ve kümeyi tanıyan sonlu bir özdevinir oluşturulabilirse, kümenin düzgün bir küme olduğu anlaşılır. Bir başka yöntem ise, kümeyi düzgün bir deyimle göstermeye çalışmaktır. Bu konu bir sonraki kesimde incelenecektir. (bkz 2.2.). Biçimsel olmayan bir başka yöntem ise, tanımlanan dizgileri tanımak için sonlu özdevinirin sahip olması gereken durum (bellek) miktarını irdelemektir. Örnek olarak P_5'i tanıyacak bir makineyi (M_5) düşünelim. Bu makinenin, dizgininin başındaki 0'ların sayısını saklaması ve 1'lerin sayısının 0'ların sayısına eşit olduğunu denetlemesi gerekir. Küme tanımında, kümedeki dizgilerin n tane 0'dan sonra n tane 1 içerdiği belirtilmekte ve n'nin birden büyük olduğu belirtilmektedir. Ancak n'nin üst sınırı yoktur. n çok büyük bir sayı olabilir. Bu nedenle, sonlu sayıda durumu bulunan, bu nedenle sonlu saklama kapasitesine sahip bir FA ile P_5'i tanımak mümkün değildir. Başka bir deyişle, P_5 kümesini tanıyan bir makine (M_5) varsa, bu makine sonlu bir özdevinir değildir. Benzer biçimde P_6'yı tanıyacak bir makinenin (M_6), dizgiyi taraması ve sürekli olarak o ana kadar rastlanan 0'larla 1'lerin sayılarının farkını saklaması gerekir. Eğer dizginin sonunda, 0'larla 1'lerin sayılarının farkı sıfırsa, dizgi M_6 tarafından tanınacaktır. Ancak 0'larla 1'lerin sayılarının farkının bir üst sınırı yoktur. Bu fark, düşünülebilecek tüm sonlu sayılardan daha büyük olabilir. Bu nedenle de M_6 bir sonlu özdevinir olamaz. P_5 ve P_6 kümelerini tanıyan sonlu özdevinirler bulunmadığı için, bu kümeler düzgün kümeler değildir.

2.2.Düzgün Deyimler

Düzgün deyimler, düzgün kümeleri biçimsel olarak tanımlamak için kullanılan bir anlatım aracı, bir dildir. Her düzgün deyim, belirli bir alfabedeki simgelerden oluşturulan dizgilerin bir alt kümesini tanımlar.

Tanım 2.1.

{a, b, c, …} alfabesindeki düzgün deyimler özyineli olarak aşağıdaki gibi tanımlanır:

1. Alfabedeki her simge bir düzgün deyimdir.

a = {a},

b= {b}, …

2. λ ve θ birer düzgün deyimdir.

λ = {λ}

θ = {}

3. Eğer P ve Q birer düzgün deyimse:

P + Q **PQ** **P***

da birer düzgün deyimdir.

Burada üç tane küme işleci tanımlanmaktadır:

+ küme birleşim işlecidir: **P + Q = P∪Q**

.(ya da boşluk) art arda ekleme (concatenation) işlecidir:

$$PQ = \{\widehat{pq} \mid p \in P, q \in Q\}$$

* kapanış (c 'PP+PPPP+…

4. Yukarıdaki kuralların s turulan deyimler düzgün deyimlerdir.

Örnekler:

Aşağıda {a, b, c, d} ve {0, 1} alfabelerinde oluşturulmuş düzgün deyim örnekleri ve bunlara karşı gelen kümeler görülmektedir.

Düzgün Deyim	Tanımladığı Küme
Φ	{}
λ	{λ}
00 + 11	{00, 11}
a(b + c)	{ab, ac}
ab*	{a, ab, abb, abbb, abbbb, …}
a(bb + cc)d*	{abb, abbd, abbdd, …, acc, accd, accdd}
a*	{λ, a, aa, aaa, aaaa, …}
(0 + 1)*	{λ, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, …}
a(b + cd*)*a	{aa, aba, acda, acdda, acddda, abca, acba, …}

2.2.1.Düzgün Deyimlere Karşı Gelen Sonlu Özdevinirlerin Bulunması

Önerme 2.1. Düzgün deyimlerle tanımlanan kümeler düzgün kümelerdir. Bu önermeye göre, bir düzgün deyim verildiğinde, bu düzgün deyimle tanımlanan kümeyi tanıyan bir sonlu özdevinir bulunabilir. Bu önerme, düzgün deyimlere karşı gelen sonlu özdevinirlerin geçiş çizeneklerinin nasıl bulunacağı gösterilerek ispat edilmeye çalışılacaktır. Bunun için de, düzgün deyimlerin özyineli tanımında yer alan her bir madde için ilgili geçiş çizeneğinin nasıl oluşturulacağı gösterilecektir.

$\mathbf{P_1}$ = a_i düzgün deyimi ile tanımlanan küme tek bir giriş simgesi içermektedir. Bu kümeyi tanıyan sonlu özdevinirin geçiş çizeneği Çizim 2.1.a'daki gibi oluşturulabilir.

$\mathbf{P_2}$ = λ düzgün deyimi ile tanımlanan küme yalnız boş simgeyi (λ) içeren kümedir. Bu kümeyi tanıyan sonlu özdevinirin geçiş çizeneği Çizim 2.1.b'deki gibi oluşturulabilir.

$\mathbf{P_3 = \theta}$ düzgün deyimi ile tanımlanan küme boş kümedir. Bu kümeyi tanıyan sonlu özdevinirin geçiş çizeneği Çizim 2.1.c'deki gibi oluşturulabilir.

P ve Q birer düzgün deyim olsun bu düzgün deyimlerle tanımlanan kümeleri tanıyan geçiş çizeneklerinin bilindiğini varsayalım. Eğer P ve Q'yu tanıyan geçiş çizeneklerinde birden çok başlangıç ve/ ya da uç durum varsa, Çizim 2.1.d ve 2.1.e'de görüldüğü gibi bu geçiş çizeneklerini bir başlangıç durumu, bir de uç durumu bulunan eşdeğer çizeneklerle dönüştürebiliriz. Bunun için, eğer çizenekte birden çok başlangıç durumu varsa, çizeneğe yeni bir durum (q_0) eklenerek tek başlangıç durumu bu yeni durum yapılır ve bu yeni durum ile eski başlangıç durumları arasına birer λ-geçişi konulur. Benzer biçimde, eğer çizenekte birden çok uç durum varsa, çizeneğe yeni bir uç durum (q_1) eklenerek tek uç durum bu yeni durum yapılır ve eski uç durumlar ile bu yeni durum arasına birer λ-geçişi konulur.

Eğer P ve Q birer düzgün deyimse ve bu düzgün deyimleri tanıyan tek başlangıç, tek de uç durumu bulunan geçiş çizenekleri biliniyorsa;

PQ

P+Q

P*

düzgün deyimlerini tanıyan geçiş çizeneklerini Çizim 2.1.f, 2.1.g ve 2.1.h'da görüldüğü gibi oluşturabiliriz.

Yukarıdaki açıklamalar, bir düzgün deyim verildiğinde, bu düzgün deyimi tanıyan geçiş çizeneğinin sistematik biçimde oluşturulabileceği göstermektedir. Her geçiş çizeneği ile bir sonlu özdevinir (FA) tanımlandığına ve sonlu özdevinirlerin tanıdığı kümeler düzgün kümeler olduğuna göre, "düzgün deyimlerle tanımlanan kümeler düzgün kümelerdir" önermesi (Önerme 2.1) ispatlanmış olmaktadır.

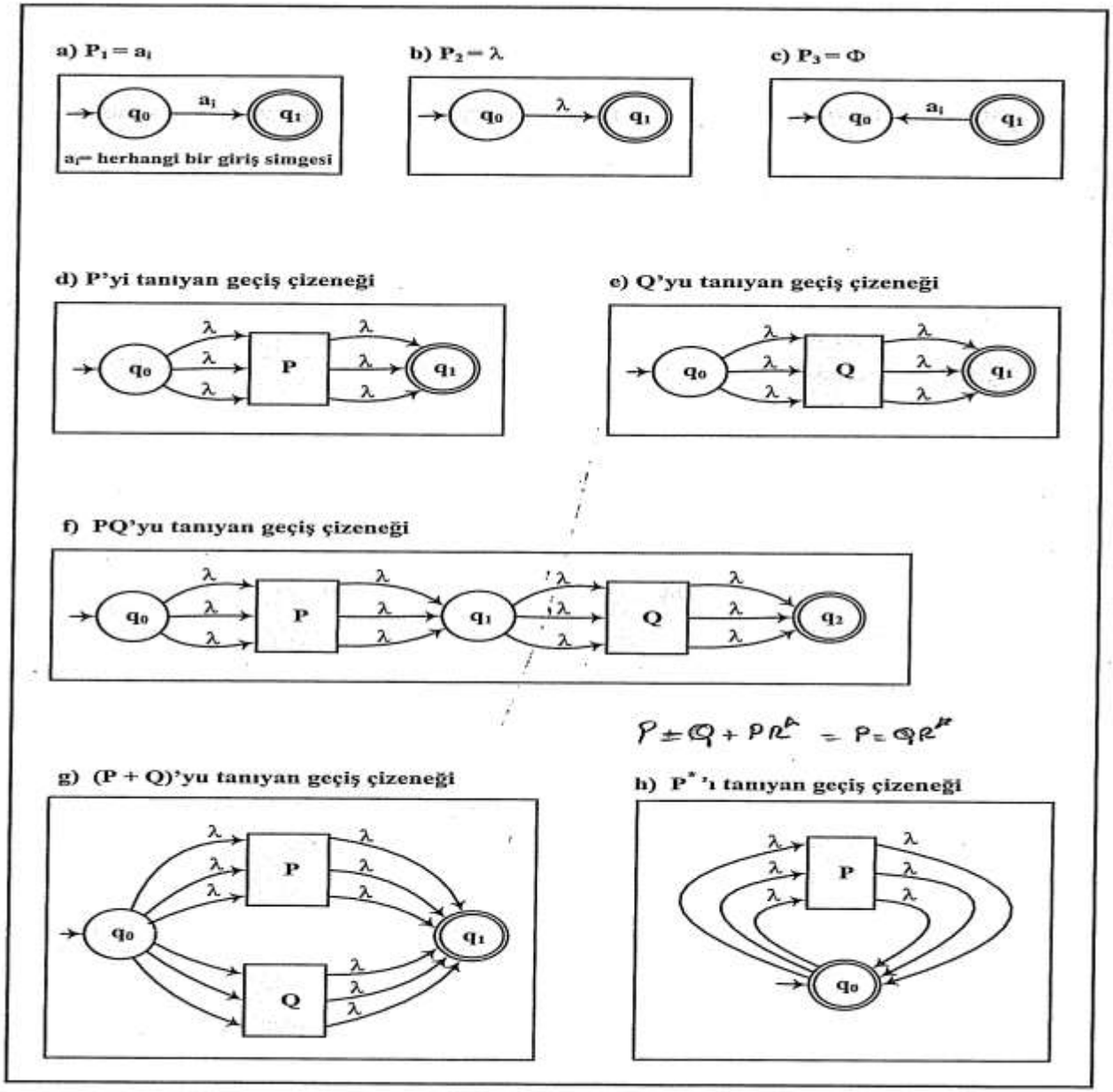
Verilen bir düzgün deyimi tanıyan geçiş çizeneğinin biçimsel yaklaşımla oluşturulması çok uzun olabilir ve bu yöntemle oluşturulan geçiş çizeneği çok sayıda λ-geçişi içerebilir. Örneğin

P = (a + bc*)*

Düzgün deyimi tanıyan sonlu özdevinirin geçiş çizeneğini biçimsel yaklaşımla oluşturmak istediğimizde,

önce a'yı, b'yi ve c'yi tanıyan çizenekleri,

sonra c*'ı tanıyan çizeneği,



Çizim 2.1. Düzgün Deyimlere Karşı Gelen Geçiş Çizeneklerinin Biçimsel Yaklaşımla Oluşturulması


POWTOON


POWTOON


POWTOON

sonra bc*'ı tanıyan çizeneği,

sonra (a + bc)*'ı tanıyan çizeneği,

son olarak da (a + bc*)*'ı tanıyan çizeneği

oluşturmak gerekir. (Çizim 2.2.a). Oysa P'yi tanıyan geçiş çizeneğini sezgisel yaklaşımla çok daha çabuk oluşturmak mümkündür. (Çizim 2.2.b). Çizimde görüldüğü gibi sezgisel yaklaşımla oluşturulan geçiş çizeneği, biçimsel yaklaşımla oluşturulan geçiş çizeneğine oranla çok daha az λ-geçişi içeren, çok daha yalın bir çizenektir.

Çizim 2.3'de düzgün deyimleri tanıyan geçiş çizeneklerinin sezgisel yaklaşımla oluşturulmasına ilişkin bir dizi örnek yer almaktadır. Geçiş çizeneklerinin sezgisel yaklaşımla oluşturulması sırasında dikkat edilmesi gereken önemli hususlardan birkaçı aşağıda yer almaktadır.

1. Eğer düzgün deyimde:

   …+a*b… ya da

   (a*b…)*

gibi bir örüntü varsa, üzerinde a döngüsü bulunan düğüme λ-geçişi ile gelinmelidir.

2. Eğer düzgün deyimde:

   …ab*+… ya da

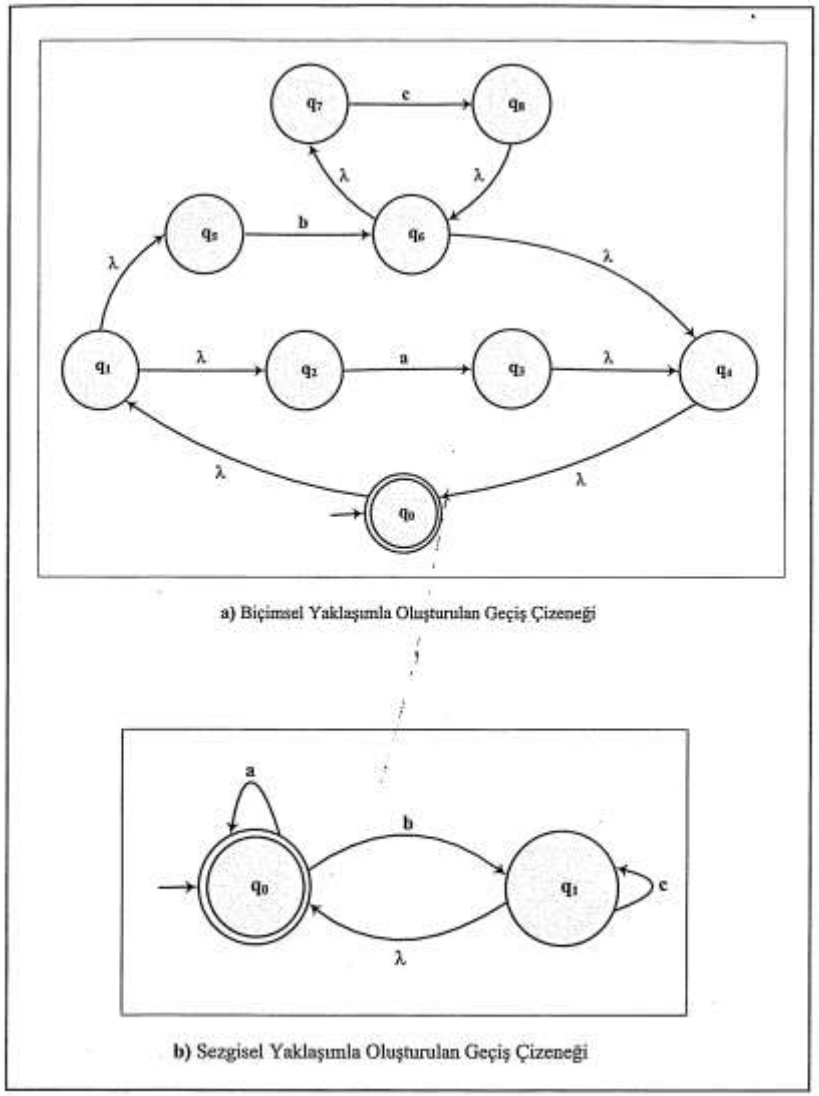
   …ab*)*

gibi bir örüntü varsa, üzerinde b döngüsü bulunan düğümden λ-geçişi ile çıkılmalıdır.

3. Eğer düzgün deyimde:

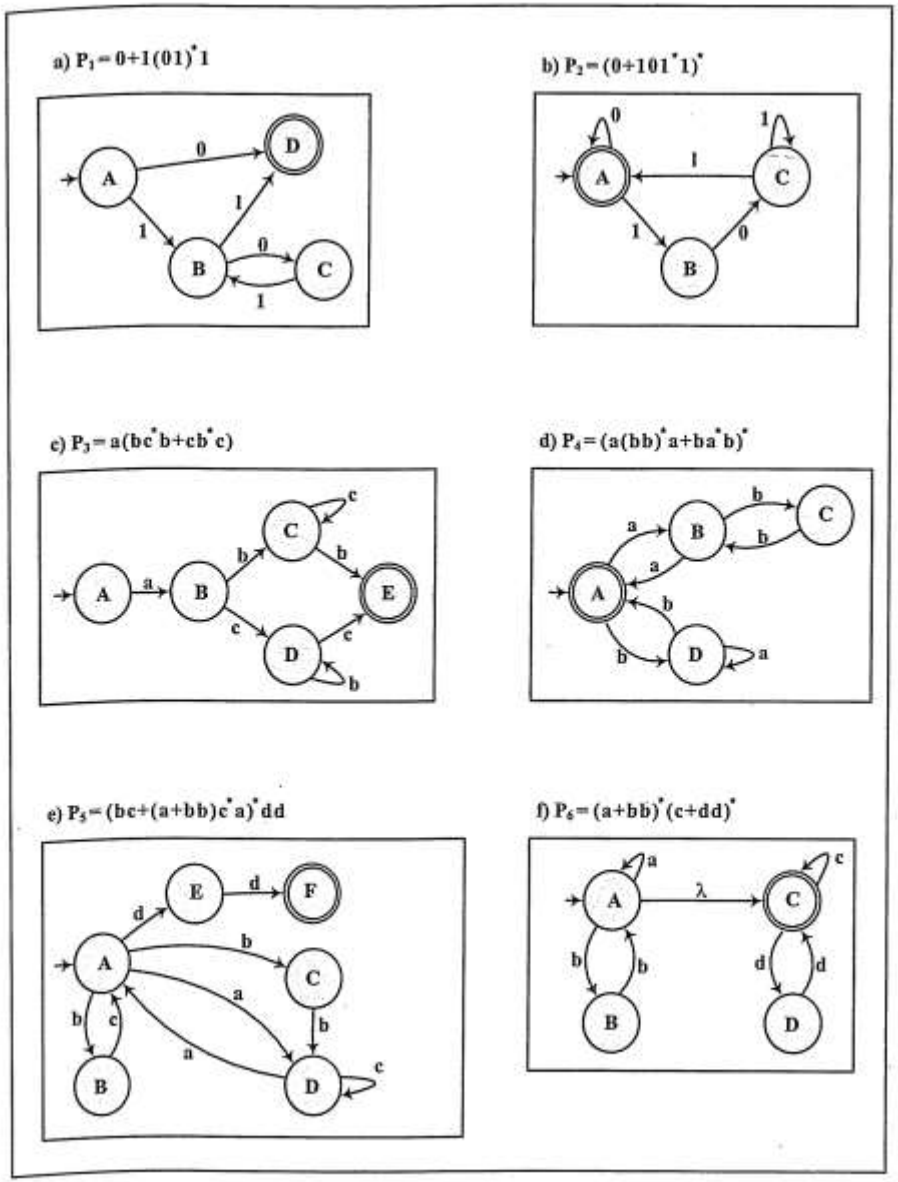
   …a*b*…

gibi bir örüntü varsa, üzerinde a döngüsü bulunan düğümden, üzerinde b döngüsü bulunan λ-geçişi ile geçilmelidir.

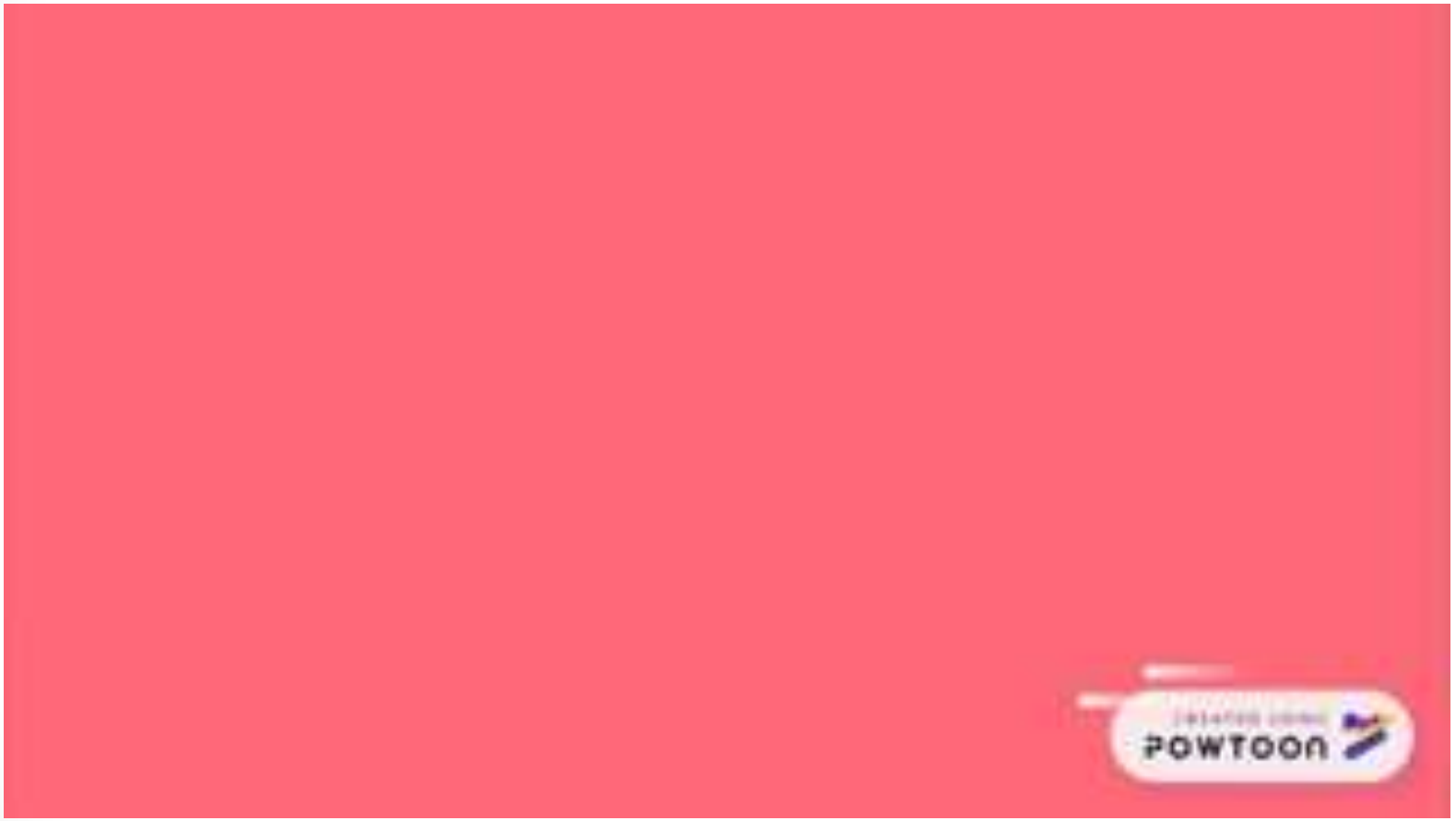


Çizim 2.2. $P = (a + bc^*)^*$ Düzgün Deyimini Tanıyan FA'nın Geçiş Çizeneğinin Biçimsel ve Sezgisel Yaklaşımla Oluşturulması


POWTOON



Çizim 2.3. Düzgün Deyimlere Karşı Gelen Geçiş Çizeneklerinin Sezgisel Yaklaşımla oluşturulması


POWTOON

2.2.2.Sonlu Özdevinirlerin Tanıdığı Kümelerin Birer Düzgün Deyim Olarak Bulunması

Önerme 2.2. Her düzgün küme bir düzgün deyimle gösterilebilir. Düzgün kümeler, sonlu özdevinirler tarafından tanınan kümeler olduğu için, bu önermeye göre verilen bir sonu özdevinirin tanıdığı düzgün kümeyi bir düzgün deyimle göstermek mümkündür.

Bu önermenin ispatı için verilen bir sonlu özdevinire karşı gelen düzgün deyimin nasıl elde edileceği gösterilecektir. Düzgün deyimin elde edilmesi için kullanılacak yöntem, denklem sistemi çözmeye dayalı olacaktır. Denklem sisteminin nasıl kurulacağı ve çözüleceğini örnekler üzerinde göstermeden önce, denklem sisteminin çözülmesinde yararlanılacak bir özelliği tanımlamak gereklidir.

Önerme 2.3.P, Q ve **R** aynı alfabede tanımlanmış düzgün deyimler ise P λ'yı içermiyorsa:

R = Q + RP

denkleminin tek çözümü R = QP*'dır.

Denklem sistemlerinin çözümünde kullanılacak bu önermenin ispatı verilmeyecektir.

Tek başlangıç durumu bulunan bir geçiş çizeneği verildiğinde her durum için aynı adı taşıyan bir değişken (küme değişkeni) tanımlanır. A durumu için tanımlanan A değişkeni, başlangıç durumundan başlayıp A durumunda son bulunan tüm dizgileri içeren bir kümeye karşı gelir. Durumlar arası geçişler dikkate alınarak her A durumu için, sol tarafında bu duruma karşı gelen değişken; sağ tarafında ise, A durumunda son bulan her

$$\delta(B, a_i) = A$$

geçişi için Bai çarpımının yer aldığı

$$A = \sum Ba_i$$

Biçiminde bir denklem kurulur. Daha sonra da Önerme 2.3'den yararlanılarak denklem sistemi çözülür ve her değişken için bir düzgün deyim elde edilir. sonlu özdevinirin tanıdığı küme, uç durumlara karşı gelen düzgün deyimlerin küme birleşimi (+) ile gösterilir.

Sonlu özdevinirlere karşı gelen düzgün deyimlerin nasıl bulunacağı iki örnek üzerinde gösterilecektir.

Örnek 2.1. Durum çizeneği Çizim 2.4.a'da görülen sonlu özdevinirin tanıdığı kümeyi bir düzgün deyim olarak bulalım. Makinenin A ve B adlı iki durumu bulunduğu için iki değişkenli aşağıdaki denklem sistemi kurulur:

$$A = \lambda + A0 + B1$$

$$B = A0$$

Denklem(1)'de B'nin yerine denklem(2)'deki A0 değeri konulduğunda:

$$A = \lambda + A0 + A01 = \lambda + A(0 + 01)$$

elde edilir. Denklem(3), Önerme 2.3 uygulanarak çözüldüğünde A'ya karşı gelen aşağıdaki düzgün deyim elde edilir:

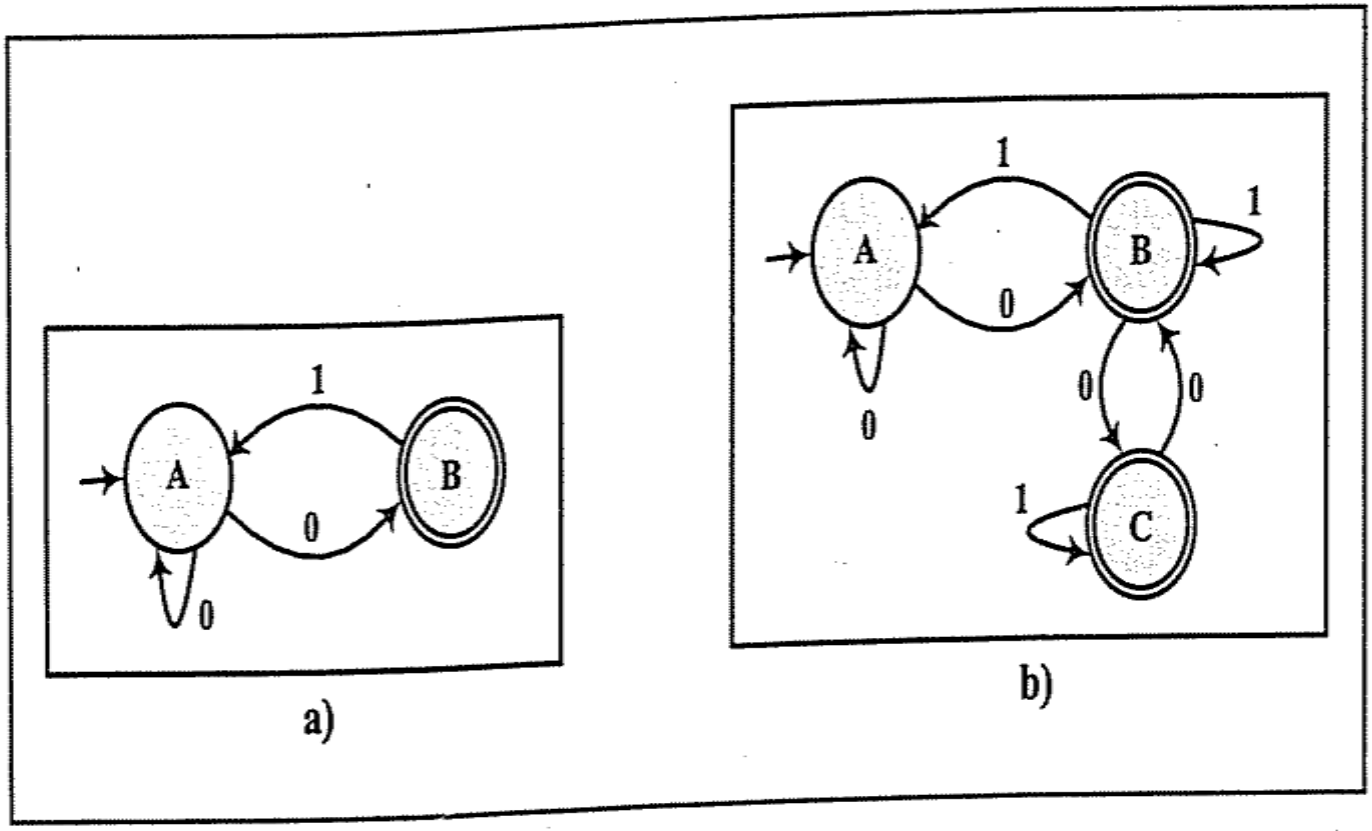
$$A = \lambda(0 + 01)^* = (0 + 01)^*$$

Denklem(2)'de A'nın yerine(4)'de elde edilen düzgün deyim konulduğunda, B'ye karşı gelen aşağıdaki düzgün deyim elde edilir:

$$B = A0 = (0 + 01)^*$$

Makinenin tek uç durumu B olduğu için aşağıdaki düzgün deyim makinenin tanıdığı kümeye karşı gelir:

$$T(M2.1) = (0 + 01)^*0$$



Çizim 2.4. Örnek 2.1 ve Örnek 2.2'deki Makinelerin Geçiş Çizenekleri


POWTOON

Örnek 2.2. Durum çizeneği Çizim 2.4.b'de görülen sonlu özdevinirin tanıdığı kümeyi bir düzgün deyim olarak bulalım. Makinenin A, B ve C adlı üç durumu bulunduğu için üç değişkenli aşağıdaki denklem sistemi kurulur:

$$A = \lambda + A0 + B1$$

$$B = 0 + B1 + C0$$

$$C = B0 + C1$$

Üçüncü denkleme Önerme 2.3 uygulandığında:

$$C = B01^*$$

elde edilir. İkinci denklemde C'nin yerine B01* değeri konularak da:

$$B = A0 + B1 + B01^*0 = A0 + B(1 + 01^*0)$$

elde edilir. birinci denklemde B'nin yerine yukarıdaki değer konulduğunda ise birinci denklem aşağıdaki biçime dönüşür:

$$A = \lambda + A0 + A0(1 + 01^*0)^*1 = \lambda + A(00(1 + 01^*0)^*1)$$

Önerme 2.3 kullanılarak yukarıdaki denklem çözüldüğünde ise A'ya karşı gelen aşağıdaki düzgün deyim elde edilir:

$$A = \lambda(0 + 0(1 + 01*0)*)1)* = (0 + 0(1 + 01*0)*1)*$$

Denklem (60)'da A'nın yerine (8)'de bulunan değeri konularak B'ye karşı gelen aşağıdaki düzgün deyim elde edilir:

$$B = (00(1 + 01*0)*1)*0(1 + 01*0)*$$

Son olarak da denklem (4)'de B'nin yerine (9)'da bulunan değeri konularak C'ye karşı gelen düzgün deyim elde edilir.

$$C=(0+0(1+01*0)*1)*0(1+01*0)*01*$$

Denklem sistemi çözülüp, her bir duruma karşı gelen düzgün deyim elde edildikten sonra, makinenin tanıdığı kümeye karşı gelen düzgün deyim aşağıdaki gibi elde edilir.

$$T(M_{2.2}) = B + C = B + B01* = B(\lambda + 01*)$$

$$T(M_{2.2}) = (0 + 0(1 + 01*0)*1)0(1 + 01*0)*(\lambda + 01*)$$

BÖLÜM 3
Dilbilgisi ve Diller

Dilbilgisi ve Diller

Her biçimsel dil belirli bir alfabe üzerinde tanımlanır. Alfabe ise sonlu sayıda simgeden oluşan bir kümedir. Alfabedeki simgelerin art arda getirilmesi ile dizgiler (*strings*) oluşturulur. Biçimsel dil, bir alfabedeki simgelerden oluşturulan dizgilerin bir kümesidir. Buna göre, bir alfabedeki simgelerden oluşturulabilecek tüm dizgileri içeren kümeyi E ile gösterirsek, bu alfabe üzerinde tanımlanan her dil E'nin bir altkümesidir. E'deki dizgilerden, dilde yer alanlar dilin tümceleridir (*sentences*). Bir alfabe üzerinde tanımlanan biçimsel bir dil, bu alfabedeki simgelerden oluşan dizgileri "geçerli" ve "geçersiz" olmak üzere ikiye ayırır. Dilde yer alan ve dilin tümcelerini oluşturan dizgiler "geçerli", dilde yer almayan dizgiler ise "geçersiz" dizgilerdir. Biçimsel dil açısından dizgi (string), tümce(*sentence*) ve sözcük(*word*) terimleri birbirinin yerine kullanılabilmektedir. Bu kitapta, dizgi ve sözcük terimleri eş anlamlı kullanılacaktır. Buna göre bir alfabe ve bu alfabe üzerinde tanımlı bir dil düşünüldüğünde, alfabedeki simgelerden oluşturulan ve dilde yer alan (geçerli) dizgiler dilin tümcelerini oluşturacaktır. Dilin hangi tümcelerden oluştuğunu gösteren kurallar bütünü ise dilbilgisi(*grammar*) olarak adlandırılacaktır.

Biçimsel dilbilgisi ve dillerin incelenmesinde, değişik harf grupları değişik alanlarda kullanılır. Harf grupları ile kullanım alanları arasındaki eşleme Çizelge 3.1'de görüldüğü gibidir.

Çizelge 3.1. Harf Grupları ve Kullanım Alanları

Harf Grubu	Örnekler	Kullanım Alanı
Latin alfabesinin başındaki büyük harfler	A, B, C, …	Sözdizim değişkenleri
Latin alfabesinin başındaki küçük harfler ve rakamlar	a, b, c, …, 0, 1,2, …	Uç simgeler
Latin alfabesinin sonundaki büyük harfler	U, V, W, X, Y, Z, …	Sözdizim değişkeni ya da uç simgeler
Yunan alfabesinin başındaki küçük harfler	α, β, γ, …	Tümcesel yapılar

3.1.Dilbilgisi ve Dilin Biçimsel Tanımı

Biçimsel olarak dilbilgisi bir dörtlü olarak tanımlanır:

$G = <V_N, V_T, P, S>$

V_N: Söz dizim değişkenleri kümesi(sonlu bir küme)

V_T: Uç simgeler kümesi (sonlu bir küme)

V_N ve V_T ayrık kümelerdir: $V_N \cap V_T = \theta$

S: Başlangıç değişkeni: $S \in V_N$

P: Yeniden yazma ya da türetme kuralları

$\alpha => \beta$

biçimindedir ve "α"nın yerine β konulabilir" diye okunur.

En genel (kısıtlamasız) biçimiyle α ve β aşağıdaki gibi tanımlanır:

$\alpha \in V^+$ $\beta \in V^*$

$V = V_N \cup V_T$ $V^+ = V^* = \{\lambda\}$

Bir dilbilgisi tarafından tanımlanan dil biçimsel olarak aşağıdaki gibi tanımlanır:

$$L(G) = \{w \mid w \in VT^*, S^* \Rightarrow w\}$$

Yukarıdaki tanıma göre, bir dilin tümceleri, başlangıç simgesinden (S'den) başlanarak ve yeniden yazma kuralları yeterli sayıda kullanılarak elde edilen uç simge dizgileridir.

$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n \Rightarrow w$ $\alpha_1, \alpha_2, \ldots, \alpha_n$: tümcesel yapılar

w : tümce

Örnek 3.1.

$G3.1 = <V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{0, 1\}$

P: S => 0S1

S => 01

G3.1 tarafından türetilen tümcelerden birkaçını bulalım:

S => 01

S => 0S1 => 0011

S => 0S1 => 00S11 => 000111

Yukarıdaki tümce örneklerinden, L(**G3.1**) dilinin aşağıdaki gibi tanımlanabileceği görülmektedir:

$L(G) = \{0^n1^n \mid n >= 1\}$

Dilbilgisi ve Dillerin Sınıflandırılması

Dilbilgisi ve türettikleri diller, yeniden yazma kurallarının özelliklerine göre:

tür-0 ya da kısıtlamasız dilbilgisi ve diller,

tür-1 ya da bağlama-bağımlı dilbilgisi ve diller,

tür-2 ya da bağlamdan-bağımsız dilbilgisi ve diller,

tür-3 ya da düzgün dilbilgisi ve diller

olmak üzere 4 sınıfa ayrılır.

3.2.1.Tür-0 Dilbilgisi ve Dil

Tür-0 ya da kısıtlamasız dilbilgisinin yeniden yazma kuralları

$$\alpha \Rightarrow \beta : \alpha \in V^{+} \quad \beta \in V^{*}$$

biçimindedir. Tür-0 dilbilgisi tarafından türetilen dile tür-0 dil denir. Tür-0 dillere özyineli sayılabilir (*recursively enumerable*) diller de denir. Kısaca "r.e." diye nitelenen özyineli sayılabilir diller için, dilin tümcelerini art arda türeten bir yordam (*procedure*) bulunabilir. r.e. diller için, bir tümce verildiğinde, bu tümcenin dilde yer alıp almadığını (dolayısıyla geçerli bir tümce olup olmadığını) belirleyen bir yordam da bulunabilir.

Örnek 3.2.

$G_{3\cdot2}=<V_T, V_N, P, S>$

$V_N = \{S, L, R, A, B, C\}$

$V_T=\{a\}$

P: S => LAaR

Aa => aaA AR => BR | C aB => Ba LB => LA aC

=> Ca LC => X

Tür-0 bir dilbilgisi olan $\mathbf{G_{3\cdot2}}$ tarafından

türetilen tümcelerden birkaçını bulalım:

S => LAaR=^ LaaAR => LaaC => LaCa => LCaa => aa S => LAaR=> LaaAR => LaaBR => LaBaR => LBaaR

=> LAaaR => LaaAaR => LaaaaAR => LaaaaC => LaaaCa => LaaCaa => LaCaaa => LCaaaa => aaaa

Yukarıda türetilen örnek tümceler ve dilbilgisinin kuralları dikkatle incelendiğinde $L(G_{3\cdot 2})$ dilinin tanımının aşağıdaki gibi olduğu bulunabilir.

$$L(G_{3J}) = \{a^i \mid i = 2^n, n\text{^}l\}$$

Tür-1 Dilbilgisi ve Dil

Tür-l dilbilgisinin yeniden yazma kuralları

$\alpha \Rightarrow \beta$: $\alpha \in V^+ \quad \beta \in V^* \mid \alpha \mid <= \mid \beta \mid$

biçimindedir. Görüldüğü gibi tür-1 dilbilgisinde, tür-0 dilbilgisine göre $\mid \alpha \mid <= \mid \beta \mid$ kısıtlaması getirilmektedir.

Tür-1 dilbilgisine bağlama-bağımlı (*context sensitive*) dilbilgisi de denir. Çünkü tür-1 dilbilgisi, yeniden yazma kurallarının tümü

$$\alpha_1 A \alpha_2 \Rightarrow \alpha_1 \beta \alpha_2 \; A \in V_N \;\; \alpha_1, \alpha_2, \beta \in V^*$$

biçiminde olan bir normal biçime (normal form) dönüştürülebilir. Bu normal biçimde α1…α2 bağlamında A'nın yerine β konulabildiği için dilbilgisi bağlama bağımlı dilbilgisi olarak adlandırılmaktadır.

Tür-1 dilbilgisi tarafından türetilen dile tür-1 ya da bağlama-bağımlı dil denir. Tür-1 dilin diğer bir adı özyineli(*recursive*) dildir. Kısaca “r.” Olarak nitelenen özyineli diller için, bir tümce verildiğinde, bu tümcenin dilde yer alıp almadığını (dolayısıyla geçerli bir tümce olup olmadığını) belirleyen bir algoritma bulunabilmektedir. Oysa tür-0 diller için tümcenin dilde yer alıp almadığını belirleyen bir yordam bulunabilmektedir. Algoritmanın her zaman sonlanan bir yordam olduğu düşünüldüğünde özyineli(r.) dillerin, özyineli sayılabilir(r.e) dillerin bir altsınıfı olduğu görülmektedir.

Örnek 3.3.

$G_{3.3} = <V_T, V_N, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{a, b, c\}$

P = S => aSAB

S => aAB

BA => AB

aA => ab

bA => bb

bB => bc

cB => cc

$G_{3.3}$ tarafından türetilen tümcelerden birkaçını bulalım:

S => aAB => abB => abc

S => aSAB => aaABAB => aabBAB => aabABB => aabbBB

=> aabbcB => aabbcc

$G_3$3 tür-1 bir dilbilgisidir. Yukarıdaki örneklerden, $L(G_{3.3})$ dilinin aşağıdaki gibi tanımlanabileceği görülmektedir:

$L(G_{3.3}) = \{ a^n b^n c^n \mid n > l\}$

Tür-2 Dilbilgisi ve Dil

Tür-2 dilbilgisinin yeniden yazma kuralları

$$A => \beta : A \in V_N \quad \beta \in V^*$$

biçimindedir. Yeniden yazma kurallarının sol tarafında tek bir değişken (A) yer almaktadır. Yeniden yazma kuralı, hangi bağlamda olursa olsun, A'nın yerine P konulabileceğini tanımlamaktadır. Bu özelliği nedeniyle tür-2 dilbilgisine bağlamdan-bağımsız *(contextfree)* dilbilgisi denir.

Tür-2 dilbilgisi tarafından türetilen dillere tür-2 ya da bağlamdan-bağımsız diller denir. Tür-2 dilbilgisi ve diller bilgisayar mühendisliği açısından büyük önem taşır. Çünkü programlama dilleri ve yazılım ürünlerinin birçok kesiminde bu model kullanılır.

Yukarıda verilen dilbilgisi örneklerinden $G_{3.1}$ tür-2 bir dilbilgisidir. Buna karşılık $G_{3.2}$ ve $G_{3.3}$ tür-2 dilbilgileri değildir.

3.2.4.Tür-3 Dilbilgisi ve Dil

Tür-3 dilbilgisinin yeniden yazma kuralları

$A \Rightarrow aB$

$A \Rightarrow a$

$A \Rightarrow \lambda$: $A,B \in V_N$, $a \in V_T$

biçimindedir. Yeniden yazma kurallarının sol tarafında tek bir değişken (A); sağ tarafında ise ya tek bir uç simge ya da bir uç simge ile bir değişken yer almaktadır. Tür-3 dilbilgisi tarafından türetilen dile tür-3 ya da düzgün (*regular*) dil denir. Daha önce örnek olarak verilen dilbilgilerinden hiçbiri tür-3 bir dilbilgisi değildir.

Örnek 3.5.

$G_{3.5} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{0, 1\}$

P: $S \Rightarrow 0S \mid 0A \mid 0 \mid \lambda$

$A \Rightarrow 0B$

$B \Rightarrow 1S$

$G_{3.5}$ tarafından türetilen tümcelerden birkaçını bulalım:

$S \Rightarrow 0S \Rightarrow 00$

$S \Rightarrow 0S \Rightarrow 00A \Rightarrow 000B \Rightarrow 0001S \Rightarrow 00010A$

$000100B \Rightarrow 0001001S \Rightarrow 00010010$

$G_{3.5}$ tür-3 bir dilbilgisidir. Dilbilgisinin kuralları ve yukarıdaki örnekler dikkatle incelendiğinde $L(G_{3.5})$ dilinin "{0, 1} alfabesinde içindeki her 1'den önce en az iki tane 0 bulunan dizgiler kümesi" olduğu görülür

Örnek 3.4.

$G_{3.4} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{a, b, c\}$

P = S => aB

A => a | aS | bAA

B => b | bS | Abb

$G_{3.4}$ tarafından türetilen tümcelerden birkaçını bulalım:

S => bA => baS => babA => baba

S => aB => abS => abbA => abbaS => abbaaB => abbaab

S => bA => bbAA => bbaSA => bbaaBA => bbaabSA

=> bbaabaBA => bbaababA => bbaababa

G3.4 tür-2 bir dilbilgisidir. Dilbilgisinin kuralları ve yukarıdaki örnekler dikkatle incelendiğinde, $L(G_{3.4})$ dilinin, { **a, b** } alfabesinde eşit sayıda **a** ve **b** içeren dizgiler kümesi olduğu görülür.

3.2.5.Sağ-Doğrusal ve Sol-Doğrusal Dilbilgisi

Yeniden yazma kuralları

$A \Rightarrow wB$

$A \Rightarrow w: A, B \in V_N, w \in V_T^*$

biçiminde olan dilbilgisine sağ-doğrusal (*right-linear*) dilbilgisi denir. Sağ-doğrusal dilbilgisinin yeniden yazma kurallarının sol tarafında bir değişken, sağ tarafında ise bir uç simgeler dizgisi ya da bir uç simgeler dizgisi ile bir değişken yer alır. Uç simgeler dizgisi sıfır uzunluğunda bir dizgi de olabilir.

Yeniden yazma kuralları

$A \Rightarrow wB$

$A \Rightarrow w: A, B \in V_N, w \in V_T^*$

biçiminde olan dilbilgisine sol-doğrusal *(left-linear)* dilbilgisi denir. Sol-doğrusal dilbilgisinin yeniden yazma kurallarının sol tarafında bir değişken, sağ tarafında ise bir uç simgeler dizgisi ya da bir değişken ile bir uç simgeler dizgisi yer alır. Uç simgeler dizgisi sıfır uzunluğunda bir dizgi de olabilir.

Sağ-doğrusal ve sol-doğrusal dilbilgileri tarafından türetilen diller düzgün dillerdir. Tür-3 dilbilgileri tarafından türetilen diller de düzgün diller olduğuna göre tür-3, sağ-doğrusal ve sol-doğrusal dilbilgileri denk dilbilgileridir. Tür-3, sağ-doğrusal ve sol-doğrusal dilbilgilerinin tümü düzgün dilbilgileri; türettikleri diller ise düzgün diller olarak adlandırılır.

Düzgün dilbilgileri için verilen ve birbirine denk olduğu belirtilen üç biçimden, tür-3 sağ-doğrusalın özel bir biçimidir. Başka bir deyişle her tür-3 dilbilgisi aynı zamanda sağ-doğrusaldır. Sağ-doğrusal her dilbilgisi de, ek değişkenler kullanılarak, kolaylıkla tür-3 dilbilgisine dönüştürülebilir. Sağ-doğrusal dilbilgisini tür-3 dilbilgisine dönüştürmek için,

$A \Rightarrow a_1 a_2 \ \ldots . \ a_n$ yerine $\begin{cases} A \Rightarrow a_1 A_1 \\ A_1 \Rightarrow a_2 A_2 \\ \ldots\ldots\ldots\ldots \\ A_{n-1} \Rightarrow a_n \end{cases}$

$A \Rightarrow a_1 a_2 \ \ldots . \ a_n B$ yerine de $\begin{cases} A \Rightarrow a_1 A_1 \\ A_1 \Rightarrow a_2 A_2 \\ \ldots\ldots\ldots\ldots \\ A_{n-1} \Rightarrow a_n B \end{cases}$

kuralları yazılır. Bu nedenle tür-3 ve sağ-doğrusal dilbilgilerinin denk olduğu açıktır. Sol-doğrusal ve sağ-doğrusal dilbilgilerinin denkliği ise bu kadar açık değildir. Ancak düzgün dilbilgisinin üç biçimi (tür-3, sağ-doğrusal ve sol-doğrusal) tarafından türetilen dillerin sonlu özdevinirler (FA) tarafından tanınan düzgün diller olduğu gösterildiğinde (Bkz. 3.3.1) dolaylı olarak bu üç biçimin eşdeğerliği de ispatlanmış olmaktadır.

Örnek 3.6.

$G_{3.6} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{0, 1\}$

$P = S \Rightarrow 0A$

$A \Rightarrow 10A \mid \lambda$

$G_{3.6}$ sağ-doğrusal bir dilbilgisidir. Bu dilbilgisi tarafından türetilen tümcelerden birkaçını bulalım:

$S \Rightarrow 0A \Rightarrow 0$

$S \Rightarrow 0A \Rightarrow 010A \Rightarrow 010$

$S \Rightarrow 0A \Rightarrow 010A \Rightarrow 01010A \Rightarrow 01010$

Yukarıdaki örneklerin incelenmesinden, L(G3.6) dilinin gibi tanımlanabileceği görülür:

$L(G_{3.6}) = 0(10)^* = (01)^*0$

$G_{3.6}$ tarafından türetilen ve yukarıdaki deyimlerle gösterilen dil, aşağıdaki sol-doğrusal ($G'_{3.6}$) ve tür-3 ($G''_{3.6}$) dilbilgileri tarafından da türetilir. Dolayısıyla $G_{3.6}$, $G'_{3.6}$ ve $G''_{3.6}$ aynı dili türeten denk dilbilgileridir.

$G'_{3.6} = <V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{0, 1\}$

P: $S \Rightarrow S10 \mid 0$

$G''_{3.6} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{0, 1\}$

P: $S \Rightarrow 0A$

$A \Rightarrow 1B \mid \lambda$

$B \Rightarrow 0A$

3.3.Düzgün Dil - Sonlu Özdevinir İlişkisi

3.3.1.Düzgün Dilbilgisinin Türettiği Dili Tanıyan Sonlu Özdevinirin Bulunması

Önerme 3.1. Bir düzgün dilbilgisi verildiğinde, bu dilbilgisinin türettiği dili tanıyan bir sonlu özdevinir bulunabilir.

Sonlu özdevinirler tarafından tanınan kümelerin düzgün kümeler olduğu tanımlanmıştı (Bkz 2.1). Önerme 3.1'de ise düzgün dilbilgileri tarafından türetilen dillerin sonlu özdevinirler tarafından tanındığı belirtilmektedir. Buradan, düzgün dilbilgileri tarafından türetilen dillerin (düzgün dillerin) düzgün kümeler olduğu sonucu çıkmaktadır.

Önerme 3.1, bir düzgün dilbilgisi verildiğinde, bu dilbilgisinin türettiği dili tanıyan sonlu özdevinirin nasıl bulunacağı gösterilerek ispat edilecektir. Düzgün dilbilgileri için üç farklı biçim olduğu için de, dilbilgisine eşdeğer sonlu özdevinirin nasıl bulunacağı tür-3, sağ-doğrusal ve sol-doğrusal dilbilgileri için ayrı ayrı gösterilecektir.

Tür-3 Dilbilgisinin Türettiği Dili Tanıyan Sonlu Özdevinirin Bulunması

Bir tür-3 dilbilgisi (G) verildiğinde, T(M) = L(G) eşitliğini sağlayan sonlu özdevinir (M) aşağıdaki gibi bulunur.

$G = <V_N, V_T, P, S>$

$M = <Q, \sum, q_0, \delta, F>$

$Q = V_N \cup \{C\}$

$\sum = V_T$

$q_0 = S$

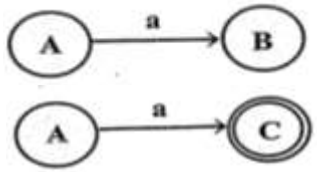
$F = \{C\}$ eğer L(G) λ'yı içermiyorsa

$= \{C, S\}$ eğer L(G) λ'yı içeriyorsa

δ: Sonlu özdevinirin durum geçişleri, dilbilgisinin yeniden yazma kurallarından aşağıdaki gibi türetilir.

$A => aB$ için $\delta(A, a) = B$

$A => a$ için $\delta(A, A) = C$



Örnek 3.7.

$G_{3.7} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{0, 1\}$

P: $S \Rightarrow 0S \mid 1A$

$A \Rightarrow 0B$

$B \Rightarrow 0B \mid 1S \mid 1$

G3.7 tür-3 bir dilbilgisidir. Bu dilbilgisinin türettiği dili tanıyan FA aşağıdaki gibi bulunur:

$M_{3.7} = <Q, \sum, q_0, \delta, F>$

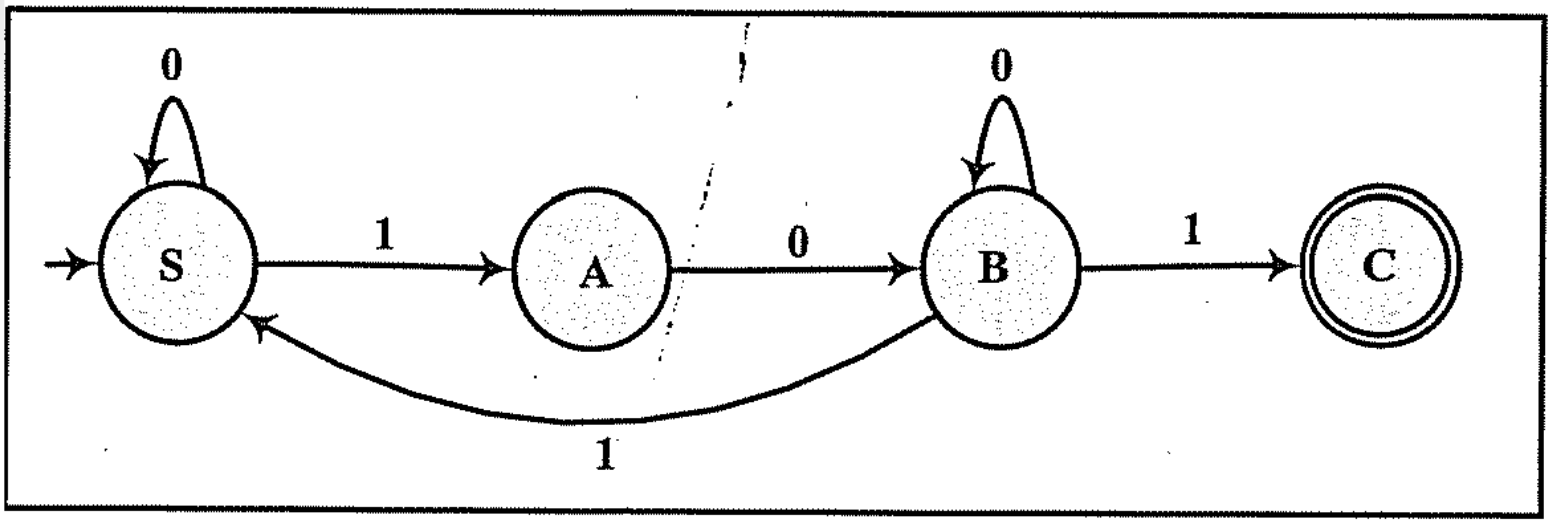
$Q = \{S, A, B, C\}$

$\sum = \{0, 1\}$

$q_0 = S$

$F = \{C\}$

δ: Makinenin durum geçişleri Çizim 3.1'de görülmektedir.



Çizim 3.1. $G_{3.7}$ 'nin türettiği Dili Tanıyan $M_{3.7}$ 'nin Geçiş Çizeneği

Sağ-Doğrusal Dilbilgisinin Türettiği Dili Tanıyan Sonlu Özdevinirin Bulunması

Bir sağ-doğrusal dilbilgisi (G) verildiğinde, T(M) = L(G) eşitliğini sağlayan sonlu özdevinir (M) aşağıdaki gibi bulunur.

$G = <V_N, V_T, P, S>$

$M = <Q, \sum, q_0, \delta, F>$

a_i : herhangi bir yeniden yazma kuralının sağ tarafının herhangi bir soneki *(suffix)*

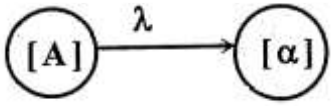
$\sum = V_T$

$q_0 = [S]$

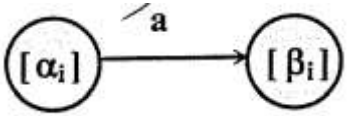
$F = [\lambda]$

δ = Sonlu özdevinirin durum geçişleri, dilbilgisinin yeniden yazma kurallarından ve bu kuralların sağ taraflarının soneklerinden aşağıdaki gibi türetilir.

Her A => α yeniden yazma kuralları için bir λ-geçişi:



İlk simgesi bir uç simge olan her $\alpha_i (\alpha_i = a\beta_i)$ için:



Örnek 3.8.

$G_{3.8} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, C\}$

$V_T = \{0, 1\}$

P: $S \Rightarrow 0S \mid 1S \mid 111A$

$A \Rightarrow 0A \mid 1A \mid \lambda$

$G_{3.8}$ sağ-doğrusal bir dilbilgisidir. Bu dilbilgisinin türettiği dili tanıyan makine ($M_{3.8}$) aşağıdaki gibi bulunur.

$Q = \{[S], [\lambda], [0S], [1S], [111A], [11A], [1A], [A], [0A]\}$

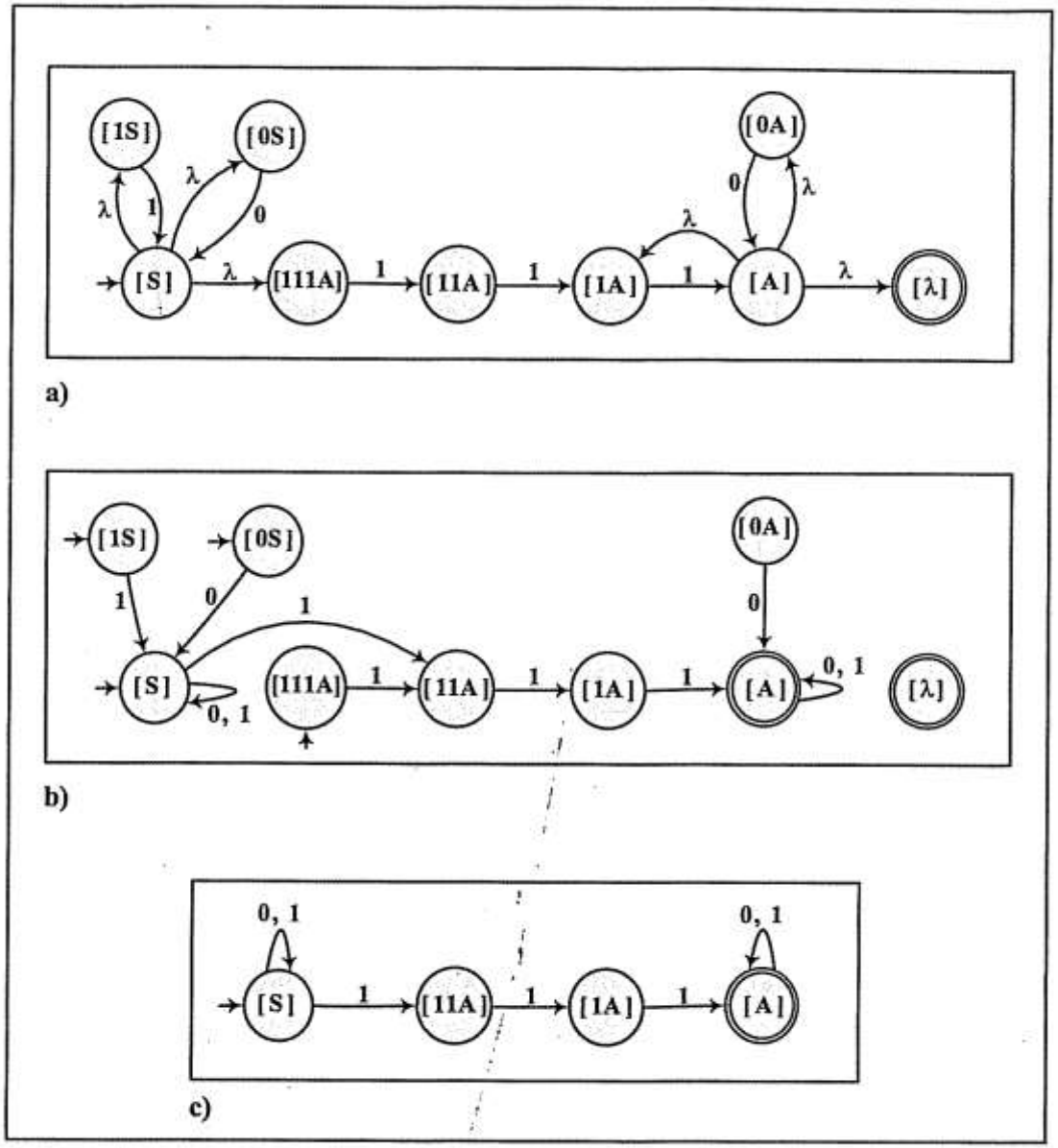
$\sum = \{0, 1\}$

$q_0 = [S]$

$F = [\lambda]$

δ : Makinenin durum geçişleri Çizim 3.2.a'da görülmektedir.

Çizim 3.2.a'daki geçiş çizeneğinde λ-geçişleri yok edildiğinde Çizim 3.2.b'deki geçiş çizeneği elde edilir. bu geçiş çizeneğinde ki gereksiz durumlar çıkarıldığında da, Çizim 3.2.c'deki geçiş çizeneği elde edilir.



Çizim 3.2. $G_{3.8}$'in Türettiği Dili Tanıyan $M_{3.8}$'in Bulunması


POWTOON

Sol-Doğrusal Dilbilgisinin Türettiği Dili Tanıyan Sonlu Özdevinirin Bulunması

Sol-doğrusal bir dilbilgisi (G) verildiğinde, bu dilbilgisinin türettiği dili tanıyan makine (M) aşağıdaki gibi bulunur.

1. Dilbilgisindeki tüm yeniden yazma kurallarının sağ tarafları tersine çevrilerek G^R elde edilir. G^R sağ-doğrusal bir dilbilgisidir.
2. G^R'yi tanıyan makine (M^R) bulunur.
3. M^R tersine çevrilerek M elde edilir. Bunun için:
   - Başlangıç durumu uç durum, uç durumlar ise başlangıç fürumu yapılır.
   - Tüm durum geçişleri tersine çevrilir.

Örnek 3.9.

$G_{3.9} = <V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{0,1\}$

$P: S => 0 \mid S10$

$G_{3.9}$ sol-doğrusal bir dilbilgisidir. Bu dilbilgisinin türettiği dili tanıyan makine ($M_{3.9}$) aşağıdaki gibi bulunur.

Önce, yeniden yazma kuralları tersine çevrilerek $G^R_{3.9}$ elde edilir.

$G^R_{3.9} = <V_N, V_T, P^R, S>$

$P^R: S => 0 \mid 01S$

Sonra $G^R_{3,9}$ 'un türettiği dili tanıyan sonlu özdevinir ($M^R_{3,9}$) bulunur.

$M^R_{3.9} = <Q^R, S, q^R_0, 8^R, F^R>$

$Q^R = \{[S], [X], [0], [01S], [1\hat{}\ \}$

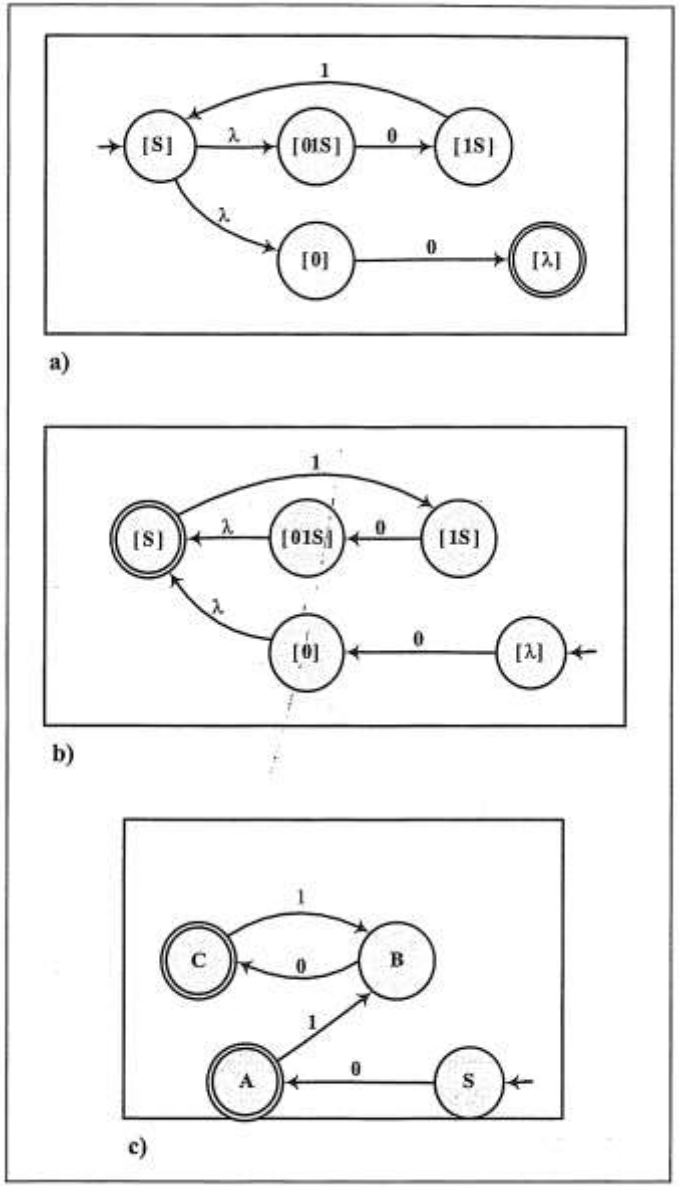
$\sum = \{0,1\}$

$q^R o = [S]$

$F^R = [X]$

S^R: Makinenin durum geçişleri Çizim 3.3.a'da görülmektedir.

Son olarak da $M^R_{3.9}$ tersine çevrilerek $M_{3.9}$ elde edilir. $M_{3.9}$ için elde edilen geçiş çizeneği Çizim 3.3.b'de görülmektedir. Bu geçiş çizeneğinde ki X- geçişleri yok edilip, yararsız durum atılıp, durumlar yeniden adlandırıldığında ise Çizim 3.3.c'de görülen geçiş çizeneği elde edilir.



Çizim 3.3. $G_{3.9}$ 'un türettiği Dili Tanıyan $M_{3.9}$ 'un Bulunması


POWTOON

3.3.2. Sonlu Özdevinirin Tanıdığı Dili Türeten Düzgün Dilbilgisinin Bulunması

Bir sonlu özdevinir (M) verildiğinde, L(G) = T(M) eşitliğini sağlayan tür-3 dilbilgisi (G) aşağıdaki gibi bulunur.

$M = <Q, \sum, q_0, 8, F>$

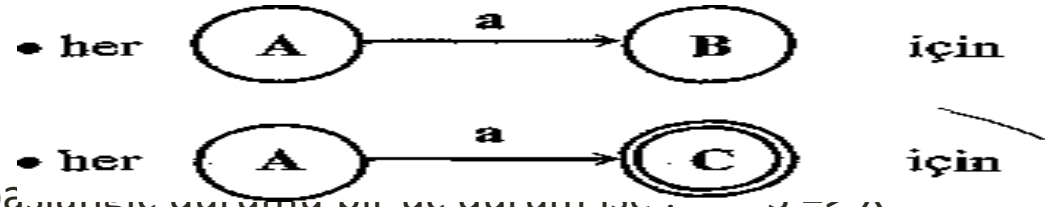
$G = <V_N, V_T, P, S>$

$V_N = Q$

$V_T = \sum$

$S = Q_0$

δ: Dilbilgisinin yeniden yazma kuralları, sonlu özdevinirin durum geçişlerinden aşağıdaki gibi türetilir.



- eğer başlangıç durumu bir aç durum ise $S \rightarrow \lambda$

Örnek 3.10.

$M_{3.10}$ = <Q, ∑, S, δ, F>

Q = {S, A, B, C, D, E}

V_T = {a, b, c, d}

F = {E}

$M_{3.10}$'nun geçiş çizeneği Çizim 3.4'de görülmektedir.

Bu makinenin tanıdığı dili türeten tür-3 dilbilgisi aşağıdaki gibi bulunur.

$G_{3.10}$ = <V_N, V_T, P, S>

V_N = Q = {S,A,B, C,D,E}

V_T = ∑ = {a, b, c, d}

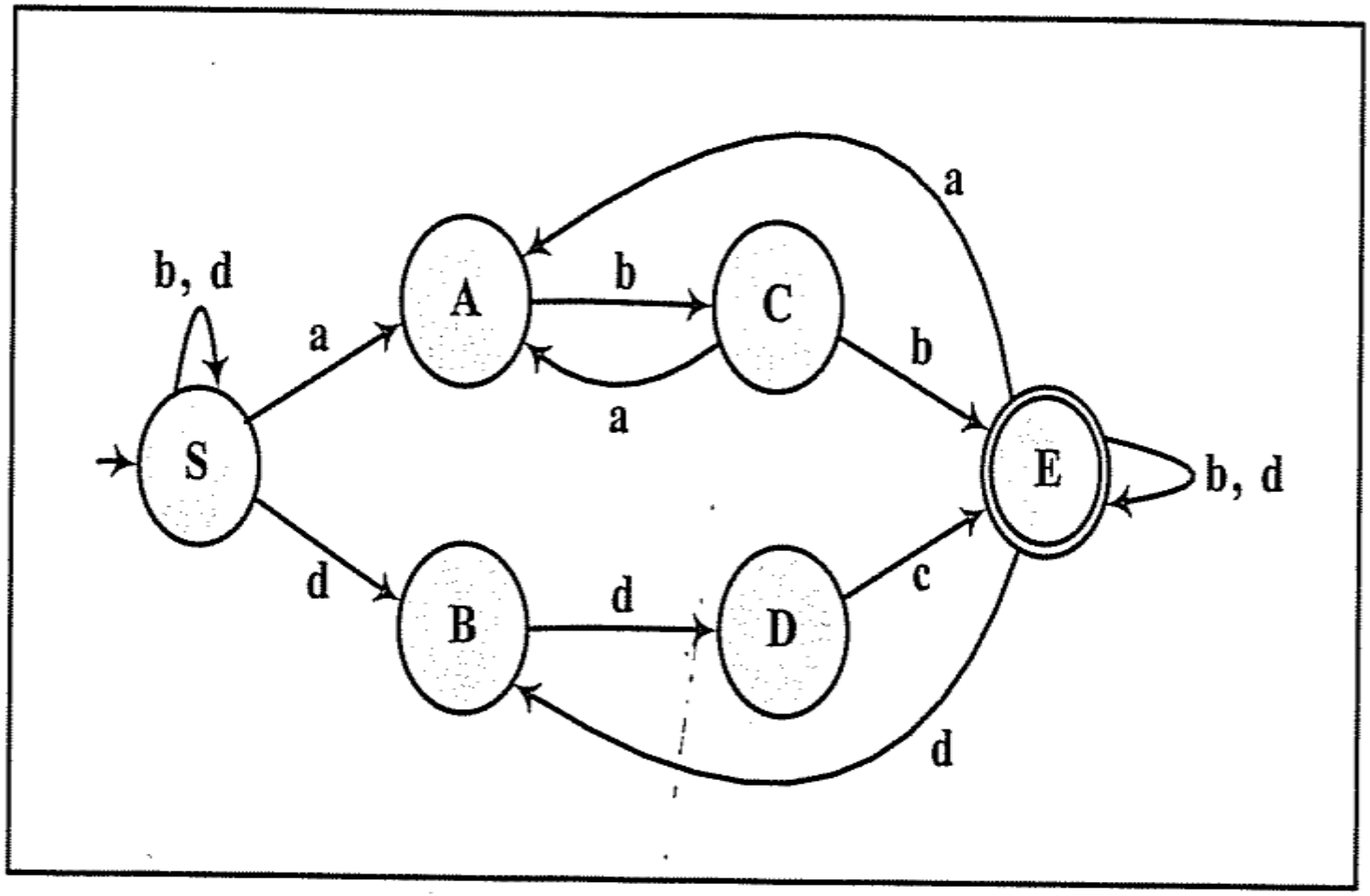
P: S => aA | bS | dS | dB

A => bC

B => dD

C => aA | bE | b

D => cE | c



Çizim 3.4. $M_{3.10}$'nun Geçiş Çizeneği

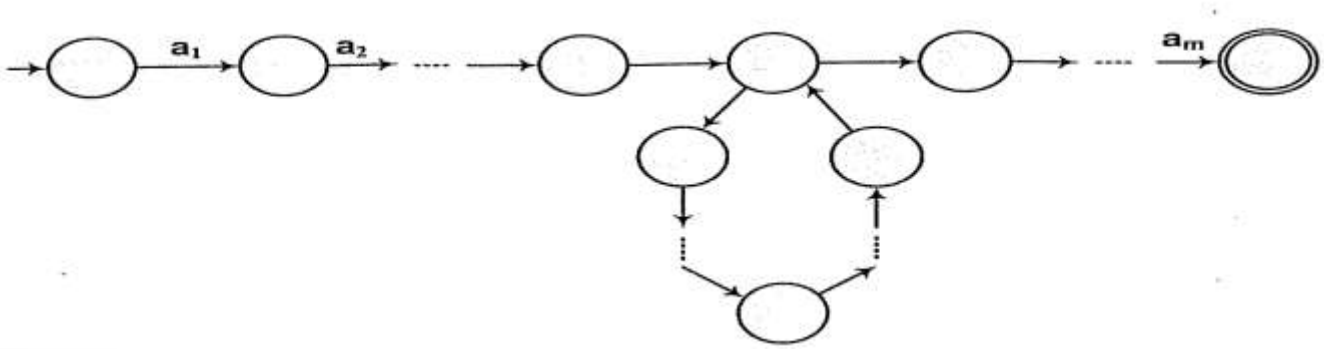

POWTOON

3.4.Düzgün Dillerin Kimi Özellikleri
Pumping Lemma

L düzgün bir dil olsun. L'yi tanıyan bir ya da kendi aralarında denk biçok sonlu özdevinir vardır. L'yi tanıyan özdevinirlerden durum sayısı en küçük olanın (M) durum sayısının n olduğunu varsayalım. Eğer L'nin sözcüklerinden birinin uzunluğu n'den büyük ise:

$$w = a_1a_2a_3\ldots.a_m \; |w| = m > n$$

M w'yi tanırken en az bir durumdan en az iki kere geçer. Başka bir deyişle, M'de w'ye karşı gelen yol mutlaka döngülüdür.



Bu durumda **M**

$w = u\,v^{i}\,t \qquad i > 0$

biçimindeki tüm tümceleri tanır ve bu tümcelerin tümü L'de yer alır. Dolayısıyla L sonsuz sayıda tümce içeren bir dildir.

Birçok dilin düzgün olmadığım ispatlamak için kullanılabilen ve *pumping lemma* olarak bilmen bu özellik şu şekilde özetlenebilir:

"Her düzgün dil için öyle bir tamsayı **(n)** vardır ki, eğer dilde uzunluğu bu değerden büyük bir tümce **(w)** varsa, bu tümce

$w = u\,v\,t$

biçiminde yazılabilir ve dil

$w = u\,v^{i}\,t \qquad i > 0$

- biçimindeki tüm tümceleri içerir".

Düzgün Dillerin Kapalılık Özellikleri

Düzgün diller küme birleşim, kesişim ve tümleme işlemlerinde kapalıdır. Buna göre eğer L_1 ve L_2 düzgün diller ise:

$L1 \cup L_2$,

$L_1 \cap L_2$,

de birer düzgün dildir.

Eğer L1 dili, $\sum$ alfabesi üzerinde tanımlı ise, L_1'in tümleri

$L_1' = \sum^* - L_1$

olarak tanımlanır.

BÖLÜM 4
BAĞLAMDAN-BAĞIMSIZ DİLBİGİSİ ve DİLLER

4.1.Bağlamdan-Bağımsız Dilbilgisi

Programlama dilleri, derleyiciler, yorumlayıcılar, söz dizim çözümleyiciler, aritmetik deyimler, ...vb. birçok yazılım bileşeninin bünyesinde yer aldığı için bağlamdan- bağımsız dilbilgisi ve diller ile bu dilleri tanıyan makine modeli bilgisayar bilimleri ve mühendisliği açısından önem taşır. •

3. bölümde tanımlandığı gibi bağlamdan-bağımsız ya da tür-2 dilbilgisi bir dörtlüdür:

$G = <V_N, V_T, P, S>$

V_N: Söz dizim değişkenleri (sonlu bir küme)

V_T: Uç simgeler kümesi (sonlu bir küme): $V_N \cap V_T = \phi$

S: Başlangıç simgesi: $S \in V_N$

P: Bağlamdan-bağımsız dilbilgisinin yazma kuralları aşağıdaki gibi tanımlanmaktadır:

$$A \Rightarrow \beta \qquad A \in VT \qquad \beta \in V^*$$

Daha önce örnek olarak verilen dilbilgilerinden $G_3.1$ ve $G_3.4$ bağlamdan-bağımsız dilbilgileridir. Aşağıda bağlamdan-bağımsız dilbilgileri için birkaç örnek daha verilmektedir.

Örnek 4.1.

$G_{4.1} = < V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{+, -, *, /, (,), v, c\}$

P: S=>S+S | S-S | S*S | S/S | (S) | v | c

$G_{4\cdot1}$ tarafından türetilen tümcelerden birkaçını bulalım.

S => S*S => S*(S) => S*(S-S) => v*(S - S)
=> v*(v - S) => v*(v - c)

S => S * S => (S)*S => (S)*(S) => (S + S)*(S)
=> (S + S)*(S - S) => (S + S)*(S – S / S)
=> (v + S)*(S – S / S) => (v + c)*(S – S / S)
=> (v + c)*(v-S/S) => (v + c)*(v-v/S)
=> (v + c) * (v – v / c)

S => S + S => S * S+ S => (S)*S + S => (S / S)*S + S
=> (S / (S))*S + S => (S / (S - S))*S + S
=> (v / (S – S))*S + S
=>(v/(v-S))*S+S => (v/(v-c))*S+S => (v/(v-c))*v + S
=> (v/(v--c))*v + c

Yukarıdaki örneklerden, bu bağlamdan-bağımsız dilbilgisi tarafından türetilen

$L(G_{4.1})$ dilinin temel aritmetik işlemler, değişkenler **(v),** değişmezler (c) ve parantezlerden oluşan geçerli aritmetik ifadeleri içeren dil olduğu görülmektedir.

Örnek 4.2.

$G_{4\cdot2} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{a, b, c, d\}$

P: S => aAdd

A => aAd | Ad | bBcc

B => bBc | Bc | λ

$G_{4\cdot2}$ tarafından türetilen tümcelerden birkaçını bulalım.

S => aAdd => abBccdd => abccdd

S => aAdd => aaAddd => aabBccddd => aabccddd

S => aAdd => aaAddd => aaAdddd => aaaAddddd

=>aaabBccddddd => aaabBccddddd

=>aaabbBccccddddd => aaabbccccddddd

Yukarıdaki örneklerden, bu bağlamdan-bağımsız dilbilgisi tarafından türetilen

$L(G_{4.2})$ dilinin aşağıdaki gibi tanımlanabileceği görülmektedir.

$L(G_{4.2}) = \{a^n b^m c^k d^p \mid n, m, p >=1, p>n, k>m\}$

Örnek 4.3.

$G_{4.3} = <V_N, V_T, P, S>$

$V_T = \{S, X, Y, Z\}$

P: $S \Rightarrow XY$

$X \Rightarrow aXbb \mid aZbb \mid abb$

$Y \Rightarrow Cy \mid c$

$Z \Rightarrow Zb \mid Xb$

Yukarıdaki örneklerden, bu bağlamdan-bağımsız dilbilgisi tarafından üretilen $L(G_{4.3})$ dilinin aşağıdaki gibi tanımlanabileceği görülmektedir.

$$L(G_{4.3}) = \{a^n b^m c^k \quad | \quad n >= 1, k >= 1, m >= 2n\}$$

Örnek 4.4.

$G_{4.4} = <V_N, V_T, P, S>$

$V_N = \{S, A\}$

$V_T = \{0, 1\}$

P: S=> 0S0 | 1S1 | 0A0 | 1A1

A=> 0A1 | 1A0 | 01 | 10

$G_{4.4}$ tarafından türetilen tümcelerden birkaçını bulalım:

S => 0S0 => 01A10 => 010A110 => 01010110

S => OSO => 01S10 => 011S110 => 0110A1110

=> 01100A11110 => 011001011110

S => 0S0 => 00S00 => 000S000 => 0001A1000

=> 00010A11000 => 000100A111000

=> 0001000A1111000 => 000100010111000

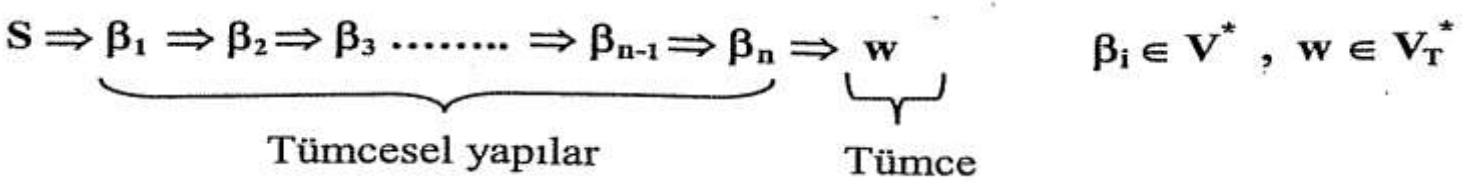
Yukarıdaki örneklerden, bu bağlamdan-bağımsız dilbilgisi tarafından türetilen **$L(G_{4\cdot4})$** dilinin aşağıdaki gibi tanımlanabileceği görülmektedir:

$L(G_{4\cdot4}) = \{U\ V\ (V')^R\ U^R\ |\mathbf{u, v \epsilon (0 + l)^*}\ ,\ |U| >= 1,\ |V| >= 1\}$

Yukarıdaki gösterimde $\mathbf{u^R}$ u'nun tersine, $(v')^R$ ise v'nin tümlerinin tersine eşittir.

4.2.Türetme Ağacı

Bir dilbilgisi tarafından türetilen tümcesel yapı ve tümceler aşağıdaki gibi gösterilebilir:

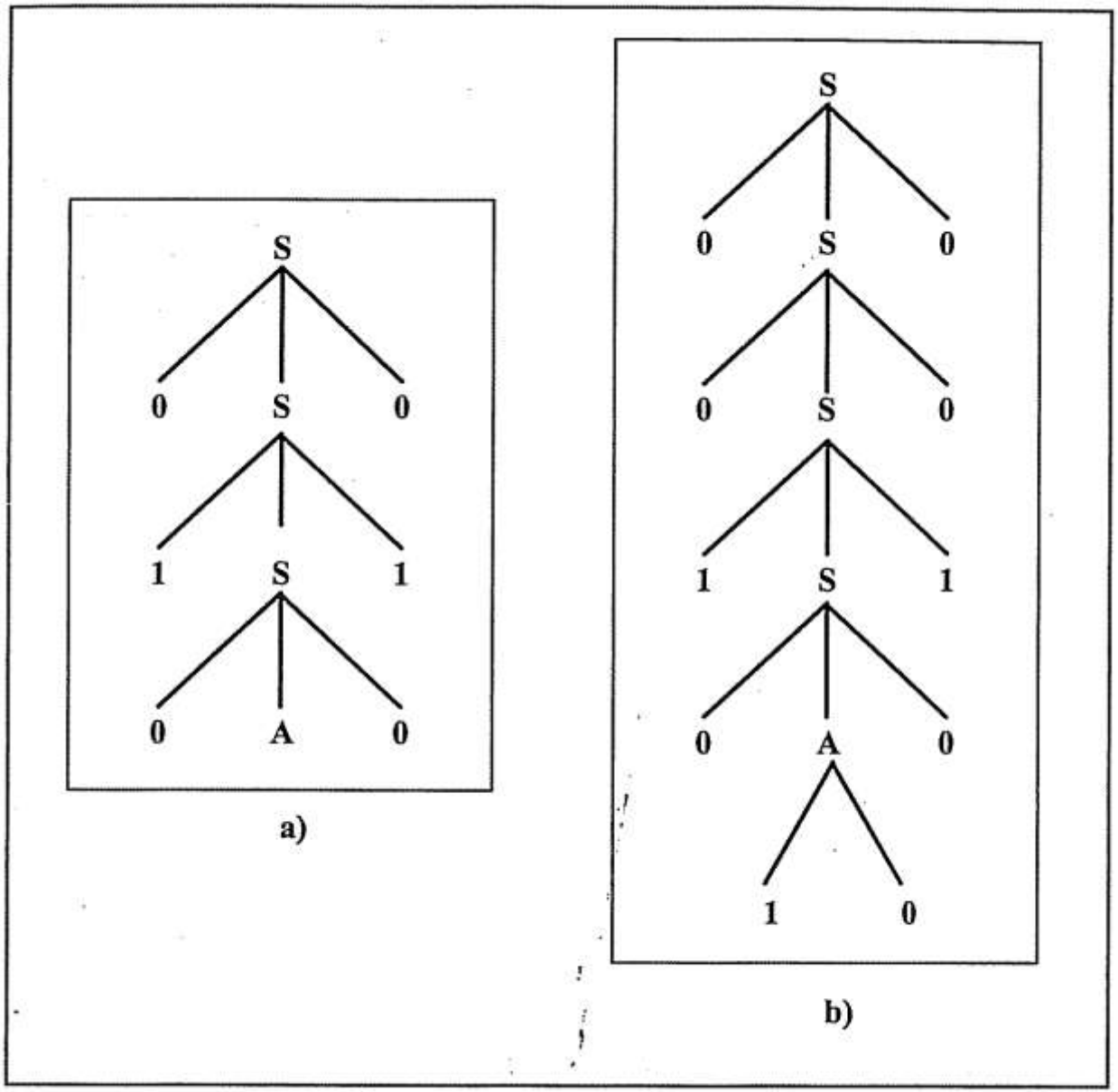


Bağlamdan-bağımsız dillerin her tümcesel yapısına ve her tümcesine bir türetme ya da ayrıştırma ağacı *(derivation or parsing tree)* karşı gelir. Örneğin $\mathbf{G_{4\cdot4}}$ tarafından türetilen:

β = 010A010 tümcesel yapısı ile

w = 0010100100 tümcesine

karşı gelen türetme ağaçlan Çizim 4.1 ’de görülmektedir.



Çizim 4.1. $G_{4.4}$ İçin Türetme Ağacı Örnekleri

Türetme ya da Ayrıştırma Ağacının Tanımı

a. Ağacın kökünün etiketi s'dir.
b. Kök dışında ara düğümlerin etiketleri söz dizim değişkenleridir. ($A \in V_N$)
c. Eğer bir tümcesel yapıya karşı geliyorsa, yaprakların etiketleri söz dizim değişkenleri ya da uç simgeler olabilir. ($X \in V$). Eğer ağaç bir tümceye karşı geliyorsa yaprakların etiketleri yalnız uç simgeler ($a \in V_N$) olabilir.
d. Eğer bir ara düğümün etiketi A, bu ara düğümün hemen altındaki düğümlerin etiketleri ise soldan sağa X_1, X_2, X_3,…, X_k ise, dilbilgisinin yeniden yazma kuralları arasında

   $A \Rightarrow X_1, X_2, X_3,\ldots, X_k \qquad X_1, X_2, X_3,\ldots, X_k \in V$

   kuralı yer almalıdır.
e. Eğer bir düğümün etiketi λ ise, bu düğümün bir uç düğüm (yaprak) olmalı ve bu düğümün kardeşi bulunmamalıdır.

Soldan ve Sağdan Türetme

Yapraklarının etiketleri uç simgeler olan her türetme ağacına dilin bir tümcesi karşı gelir. Eğer dil belirgin *(unambiguous)* bir dil ise de, dilin her tümcesine bir türetme ağacı karşı gelir. Eğer dilin bir tümcesine birden çok türetme ağacı karşı geliyorsa, dil belirgin olmayan *(ambiguous)* bir dildir. Belirginlik dilbilgisinin ve dilin en temel özelliklerinden biridir. Tümceleri birden çok anlam taşıyan belirgin olmayan dillerin uygulamada hiçbir değeri yoktur. Bu nedenle, kullanılan dillerin tümünün belirginlik özelliğini sağladığı varsayılacaktır. Buna göre dilin her tümcesine bir türetme ağacı, yapraklarının etiketleri uç simgeler olan her türetme ağacına da bir tümce karşı geldiğini söyleyebiliriz.

Dilin her tümcesine bir türetme ağacı karşı geldiği gibi, bir soldan türetme, bir de sağdan türetme karşı gelir.

Eğer bir tümce türetilirken, her adımda en soldaki değişkene bir türetme uygulanıyorsa, yapılan türetmeye soldan türetme *(leftmost derivation)* denir. Soldan türetmenin her ara adımında, eğer tümcesel yapı birden çok değişken içeriyorsa, öncelik en soldaki değişkene verilir.

Eğer bir tümce türetilirken, her adımda en sağdaki değişkene bir türetme uygulanıyorsa, yapılan türetmeye sağdan türetme *(rightmost derivation)* denir. Sağdan türetmenin her ara adımında, eğer tümcesel yapı birden çok değişken içeriyorsa, öncelik en sağdaki değişkene verilir.

Çizim4.2'de, $L(G_4.ı)$ dilinin iki tümcesine karşı gelen türetme ağacı görülmektedir. Çizimdeki türetme ağaçlarına karşı gelen tümceler sırasıyla aşağıdakilerdir.

$w_1 = (v + c)*(v - c)$

$w_2 = v / (v - c) + v*(v + c)$

Bu iki tümceye karşı gelen soldan ve sağdan türetmeler aşağıdaki gibidir.

a) w_1'in soldan türetilmesi:

S => S * S => (S)*S => (S + S)*S => (v + S)*S

=> (v + c)*S => (v + c)*(S) => (v + c)*(S-S)

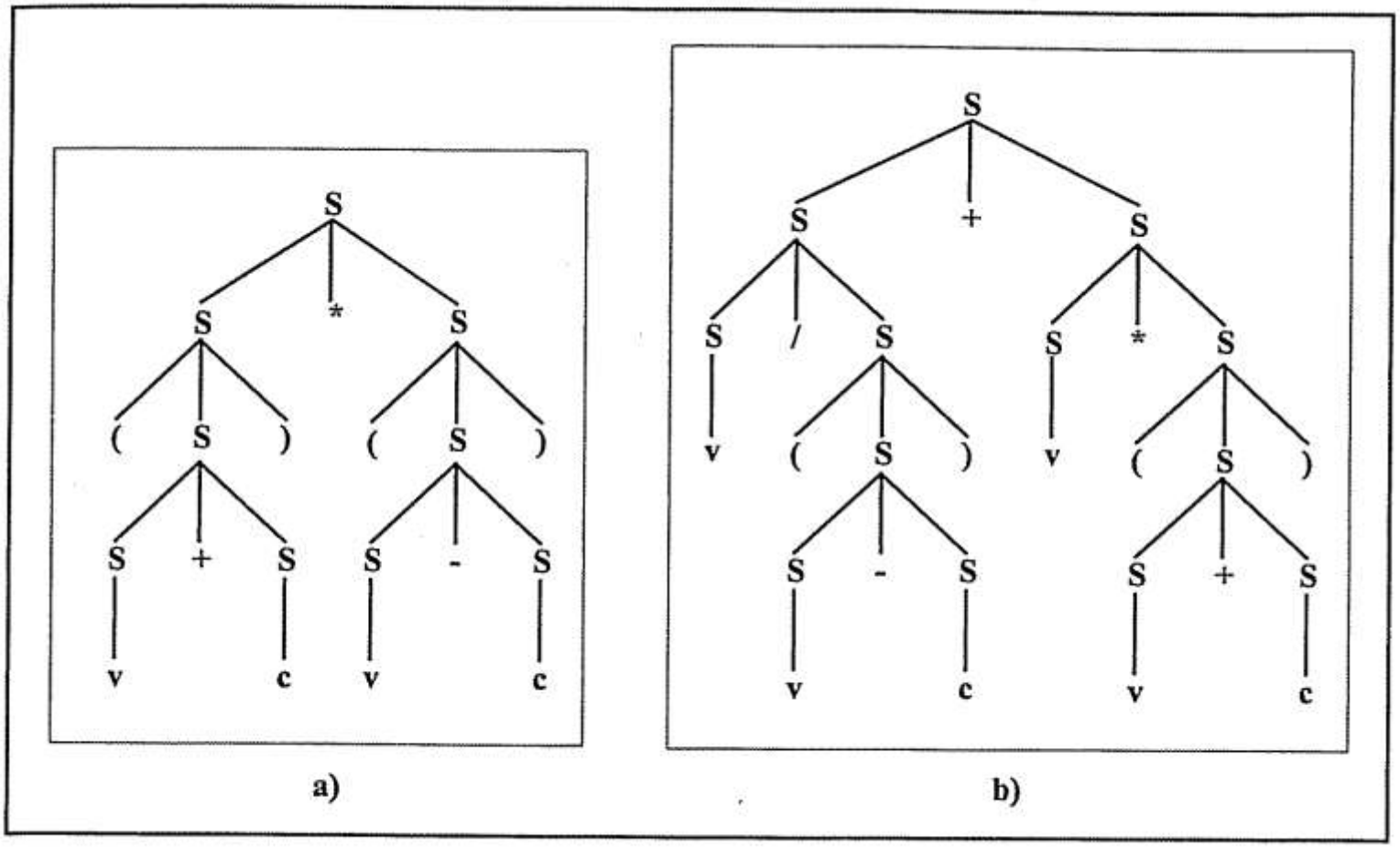
=> (v + c)*(v-S) => (v + c)*(v - c)

b) w_1'in sağdan türetilmesi:

S => S*S => S*(S) => S*(S - S) => S*(S - c)

=> S*(v-c) => (S)*(v - c) => (S + S)*(v - c)

=> (S + c)*(v - c) => (v + c)*(v-c)



Çizim 4.2. $G_{4.1}$ İçin Türetme Ağacı Örnekleri

c) w_2'in soldan türetilmesi:

S => S+ S => S/S+S => v/S+S => v/(S)+S

=> v/(S – S)+S => V/(v – S)+S => v/(v-c)+S

=> v/(v – c) +S*S => v/(v – c)+v*S

=> v/(v – c)+v*(S) => v/(v – c)+v*(S+S)

=> v/(v – c)+v*(v + S) => v/(v – c)+v*(v + c)

d) w_2'in sağdan türetilmesi:

S => S + S => S + S * S => S + S * (S) => S +* (S + S)

=> S + S*(S + c) => S + S*(v + c) => S + v*(v + c)

=> S/S + v*(v + c) => S/(S) + v*(v + c)

=> S/(S – S) + v*(v + c) => S/(S – c) + v*(v + c)

=>S/(v – c)+v*(v + c) => v/(v – c)+v*(v + c)

4.3.Dilbilgisinin Yalınlaştırılması

Bağlamdan-bağımsız bir dilbilgisi verildiğinde, bu dilbilgisinin daha az sayıda değişken ve kural içeren ve belirli biçimdeki kuralları içermeyen bir dilbilgisine dönüştürülmesine dilbilgisinin yalınlaştırması denilir. Aşağıda dilbilgisinin yalınlaştırılması ile ilgili bir dizi tanım ve algoritma verilmektedir.

4.3.1.Özyineli Kural, Özyineli Değişken

Bir yeniden yazma kuralının solundaki değişken, kuralın sağında da yer alıyorsa, başka bir deyişle kural **(A => α_1 A α_2)** biçiminde ise, bu kurala **özyineli kural** *(recursive rule)*, **A** değişkenine de **Özyineli değişken** *(recursive variable)* denir.

Eğer bir yeniden yazma kuralının solundaki değişken, aynı zamanda kuralın sağındaki ilk simge ise, başka bir deyişle kural **(A => Aα_2)** biçiminde ise, bu kurala **doğrudan özyineli kural** *(direct recursive rule}* denir.

Bazı işlemlerde dilbilgisinin başlangıç değişkeninin (S) özyineli olması istenmez. Dilbilgisine yeni bir değişken (S') ve yeni bn kural (S' => S) ekleyip, yeni eklenen değişkeni başlangıç değişkeni yaparak dilbilgisinin bu özelliği taşıması sağlanabilir.

4.3.2.Yok Edilebilir Değişken

Eğer bir değişkenin yerine bir ya da birkaç türetme sonunda X konulabiliyorsa,

A => λ ya da **A*** => λ

bu değişkene yok edilebilir *(nullable}* değişken denir. Bazı işlemlerde, başlangıç değişkeni dışında dilbilgisinde yok edilebilir değişken bulunmaması, başka bir deyişle (S => λ) dışında λ-kuralının bulunmaması istenir. Buna göre, eğer dil λ'yı içermiyorsa (λ ∉ **L),** dilbilgisinde hiç λ-kuralı bulunmayacaktır. Eğer dil λ'yı içeriyorsa da (λ ∈ **L),** dilbilgisindeki tek λ-kuralı (S => λ) olacaktır.

Bir algoritmayla, dilbilgisinin yok edilebilir değişkenlerinin hangileri olduğu kolaylıkla bulunabilir. Bir başka algoritmayla da dilbilgisi, başlangıç değişkeni dışında yok edilebilir değişkeni bulunmayan eşdeğer bir dilbilgisine dönüştürülebilir. Bu algoritmalar aşağıda verilmektedir.

Algoritma 4.1. Yok edilebilir değişkenlerin bulunması

1. $T_L = \{A \mid (A \Rightarrow \lambda) \in P$
2. $T_{Eski} = \varphi$
3. $(T_L = T_{Eski})$ oluncaya kadar aşağıdaki işlemleri tekrarla:
    1. $T_L = T_{Eski}$;
    2. V_N'deki her değişken (A) için aşağıdaki işlemleri tekrarla:

        Eğer en az bir $(A \Rightarrow \alpha)$ kuralı için $\alpha \in (T_L)'$ ise

        A'yı T_L'ye ekle;

    Son 3.2;

son 3;

Algoritma4.2. Başlangıç değişkeni dışında yok edilebilir değişkeni bulunmayan eşdeğer dilbilgisinin bulunması

1. P'deki tüm (A => λ) kurallarını çıkar;
2. Eğer λ ϵ L(G) ise P'ye (S => λ) kuralını ekle;
3. P'deki her (A => α) kuralı için aşağıdaki işlemleri tekrarla:

   3.1.Eğer α yok edilebilir değişkenler içeriyorsa, bu değişkenlerin bir ya da birkaçını α'dan çıkararak oluşturulabilen her α_i için aşağıdaki işlemleri tekrarla:

      P'ye (A => α_i)kuralını ekle;

   son 3.1;

son 3;

Örnek 4.5.

$G_{4.5.} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, C\}$

$V_T = \{a, b\}$

$P = S \Rightarrow ACA$

$A \Rightarrow B \mid C \mid aAa$

$B \Rightarrow bB \mid b$

$C \Rightarrow Cc \mid \lambda$

$G_{4.5}$'e eşdeğer, başlangıç değişkeni dışında yok edilebilir değişken içermeyen bir dilbilgisi bulmak için önce dilbilgisinin yok edilebilir değişkenlerini bulalım. Algoritma kullanılarak bu değişkenler aşağıdaki gibi bulunur:

$T_L = \{C, A, S\}$

Dilbilgisinin başlangıç değişkeni yok edilebilir bir değişken dilbilgisinin türettiği dil λ'yı içermektedir. Bu nedenle eşdeğer **(S => λ)** kuralı yer alacaktıt. Algoritma 4.2 kullanılarak, $G_{4.5}$'e eşdeğer, başlangıç değişkeni dışında yok edilebilir değişken içermeyen dilbilgisi aşağıdaki gibi bulunur:

$G'_{4.5} = < V_N, V_T, P, S >$

$V_N = \{S, A, B, C\}$

$V_T = \{a, b\}$

P: S => ACA | AC | CA | AA | A | C | λ

A => B | C | aAa] aa

B => bB | b

C => cC | c

4.3.3.Birim Türetme Kuralları

Dilbilgisindeki yeniden yazma kurallarından **(A => B)** biçiminde olanlara "birim türetme kuralı *(ımit production rule)"* ya da "zincir kural *(chain rule)"* denir. Birim türetme kuralları tümcelerin türetilmesini geciktiren kurallardır. Dilbilgisinin kullanım amacı da dilin tümcelerini türetmek olduğuna göre, birim türetme kuralları içermeyen dilbilgilerinin kullanılması daha uygundur.

Bağlamdan-bağımsız bir dilbilgisi verildiğinde, bu dilbilgisine eşdeğer, birim türetme kuralları içermeyen bir dilbilgisi bulunabilir. Bu amaçla önce dilbilgisindeki her değişken (A) için, bu değişkenden (üretilebilen değişkenler kümesinin **(T_A)** bulunması gerekir.

$$\sqrt{A} \in V_N \Rightarrow T_A = \{B \mid B \in V_N, A^* \Rightarrow B\}$$

Her değişkenden (A) türetilebilecek değişkenler kümesi Algoritma 4.3 ile bulunabilir.

Algoritma4.3.Bir değişkenden (A) türetilebilen değişkenler kümesinin bulunması

1. $T_A = \{A\}$;
2. $T_{Eski} = \varphi$;
3. ($T_A = T_{Eski}$) oluncaya kadar aşağıdaki işlemleri tekrarla:
    1. $T_{Yeni} = T_A - T_{Eski}$;
    2. $T_{Eski} = T_A$;
    3. T_{Yeni}'deki her değişken (B) için aşağıdaki işlemleri tekrarla:

        $T_A = T_A \cup \{C\}$

    Son 3.3.1;

Son3.3;

Son 3;

Her değişken için bu değişkenden türetilebilen değişkenler kümesi bulunduktan sonra, Algoritma 4.4'ü kullanarak birim türetme kuralı içermeyen eşdeğer dilbilgisi bulunabilir.

Algoritma 4.4. Birim türetme kuralı içermeyen dilbilgisinin bulunması:

1. $P = P - \{A \Rightarrow B \mid A, B \in V_N\}$
2. V_N'deki her değişken (A) için eğer T_A'da A'dan başka en az bir değişken varsa aşağıdaki işlemleri tekrarla:
   1. $T_A = T_A - \{A\}$;
   2. T_A'daki her değişken {B} için aşağıdaki işlemleri tekrarla:

      B kurallarının tümü ($B \Rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_k$)

      $P = P \cup \{A \Rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_k \}$;

   Son 2.2;

Son 2;

Örnek 4. 6.

$G_{4\cdot6} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, C\}$

$V_T=\{a, b, c\}$

P: S => A | AA | AC | CA | ACA | λ

A => B | aAa | aa

B => C | bB | b

C => cC | c

$G_{4\cdot6}$'ya eşdeğer, birim türetme kuralı içermeyen bir dilbilgisi bulmak için önce S, A, B ve C değişkenlerinden türetilebilen değişken kümelerinin bulunması gerekir. Algoritma 4.3 kullanılarak bu kümeler aşağıdaki gibi bulunur:

$T_5 = \{S, A, B, C\}$

$T_A=\{A, B, C\}$

$T_B = \{B,C\}$

$T_C= \{C\}$

Daha sonra Algoritma 4.4 kullanılarak $\mathbf{G_{4\cdot6}}$ 'ya eşdeğer aşağıdaki dilbilgisi bulunur:

$G'_{4\cdot6} = < V_N, V_T, P, S >$

$V_N = \{S, A, B, C\}$

$V_T = \{a, b, c\}$

P: S => AA [AC | CA | ACA | aAa | aa | bB | b

cC | c | λ

A => aAa | aa | bB | b | cC | c

B => bB | b | cC | c

C => cC | c

4.3.4.Yararsız Değişken, Simge ve Kurallar

Eğer bir dilbilgisinin uç simgelerinden biri, bu dilbilgisi tarafından türetilen tümcelerden hiçbirinde yer almıyorsa, bu uç simgeye yararsız simge *(useless symbol)* denilir.

Eğer bir dilbilgisi değişkenlerinden biri (A), en az bir tümce türetilirken elde edilen tümcesel yapıların en az birinde yer alıyorsa:

$$S^* \Rightarrow \alpha_1 A \alpha_2 \Rightarrow^* w$$

Bu değişken yararlı *(useful*) bir değişkendir. Bu özelliği taşımayan değişkenlere ise yararsız değişken (*useless variable*) denir. Yararsız değişken hiçbir tümcenin türetilmesinde kullanılmayan; kullanıldığı türetmelerin hiçbirinin sonunda ise bir tümce elde edilemeyen değişkendir.

Benzer biçimde, eğer bir dilbilgisinin yeniden yazma kurallarından biri ($A \Rightarrow \beta$), en az bir tümcenin türetilmesi sırasında kullanılıyorsa, bu kural yararlı bir kuraldır(*usefull rule*). Hiçbir tümcenin türetilmesinde kullanılmayan kurala ise yararsız kurallar (*useless rule*) adı verilir.

Benzer biçimde, eğer bir dilbilgisinin yeniden yazma kurallarından biri ($A \Rightarrow \beta$), en az bir tümcenin türetilmesi sırasında kullanılıyorsa, bu kural yararlı bir kuraldır(*usefull rule*). Hiçbir tümcenin türetilmesinde kullanılmayan kurala ise yararsız kurallar (*useless rule*) adı verilir.

Eğer, bağlamdan-bağımsız bir dilbilgisi yararsız değişken, uç simge ya da kural içeriyorsa, bu dilbilgisine eşdeğer olan ve yararsız uç simge, değişken ve kural içermeyen bir dilbilgisi bulunabilir. Bunun içinde öncelikle değişkenlerden hangilerinin yararlı olduğunun bulunması gerekir. Bir değişken (A) yararlı bir değişken olabilmesi için:

Bu değişkenden başlanarak en az bir uç simge dizgisinin türetilebilmesi:

$A^* \Rightarrow u \; u \in V_T^*$ (u: dilin bir tümcesi olması şart değil)

Başlangıç değişkeninden bu değişkene ulaşmanın mümkün olması:

$$S^* \Rightarrow \alpha_1 A \alpha_2$$

gereklidir. Buna göre bağlamdan-bağımsız bir dilbilgisi(G) verildiğinde, yararsız değişken ve kurallar atılarak eşdeğer bir dilbilgisinin bulunması iki adımda gerçekleştirilir.

Birinci adımda, Algoritma 4.5 kullanılarak uç simge dizgisi türeten değişkenler kümesi bulunur. Bu kümede yer almayan değişkenler yararsız değişkenlerdir. Dilbilgisinde bu değişkenler ile bu değişkenlerin yer aldığı kurallar çıkarılarak eşdeğer bir dilbilgisi (G') bulunur.

İkinci adımda, Algoritma 4.6 kullanılarak, birinci adımda bulunan dilbilgisindeki (G') değişkenlerinden hangilerine başlangıç değişkeninde ulaşılabildiği bulunur. Başlangıç değişkeninden ulaşılamayan değişkenler de yararsız değişkenlerdir. Dilbilgisinden (G') bu değişkenler ile bu değişkenlerin yer aldığı kurallar çıkarılarak eşdeğer dilbilgisi (G'') bulunur.

Aşağıda uç simge dizgisi türeten değişkenler kümesi (T_D) ile başlangıç değişkeninden ulaşılabilen değişkenler kümesi (T_U) bulmayı sağlayan algoritmalar (sırasıyla Algoritma 4.5 ve Algoritma 4.6) sunulmaktadır.

Algoritma 4.5. Uç simge dizgisi türeten değişkenler kümesinin bulunması.

1. $T_D = \{A \mid (A \Rightarrow w) \in P \text{ ve } w \in V_T^*$
2. ($T = T_{Eski}$) oluncaya kadar aşağıdaki işlemleri tekrarla:
    1. $T_{Eski} = T_D$;
    2. Dilbilgisindeki her değişken (A) için aşağıdaki işlemleri tekrarla:
        1. Her ($A \Rightarrow \beta$) kuralı için aşağıdaki işlemleri tekrarla:

            eğer $\beta \in (T_{Esk}i \cup V_T)^*$ ise

            A'yı T_D'ye ekle;

        Son 2.2.1;

    Son 2.2;

Son 2;

Algoritma 4.6. S'den ulaşılabilen değişkenler kümesinin bulunması

1. $T_U = \{S\}$;
2. $T_{Eski} = \varphi$;
3. ($T_U = T_{Eski}$) oluncaya kadar aşağıdaki işlemleri tekrarla:
    1. $T_{Yeni} = T_U - T_{esk}i$;
    2. $T_{Eski} = T_U$;
    3. T_{Yeni}'deki her değişken için aşağıdaki işlemleri tekrarla:
        1. Her (A => β) yeniden yazma kuralı için aşağıdaki işlemleri tekrarla:
            β'daki tüm değişkenleri T_U'ya ekle;
        Son 3.3.1;
    Son 3.3;
Son 3;

Örnek 4.7.

$G_{4.7} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, C, D, E, F\}$

$V_T = \{a, b\}$

P = S => B | AC | BS

A => aA | Af

B => CF | b

C => Cc | D

D => C | aD | BD

E => Aa | BSA

F => bB | b

$G_{4.7}$'ye eşdeğer; yararsız değişken, simge ve kural içermeyen bir dilbilgisi bulmak için önce $G_{4.7}$ 'deki değişkenlerden hangilerinin uç simge dizgisi türeten değişkenler olduğunu bulmak gerekir. Uç simge dizgisi türeten değişkenler kümesi Algoritma 4.5 ile aşağıdaki gibi bulunur:

$$T_D = \{S, A, B, E, F\}$$

Yukarıdaki kümede yer almayan C ve D değişkenleri yararsız değişkenlerdir. Dilbilgisinden bu değişkenler ile bu değişkenlerin yer aldığı kuralları çıkaralım:

$G'_{4.7} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, E, F\}$

$V_T = \{a, b\}$

$P = S \Rightarrow B \mid BS$

$A \Rightarrow aA \mid aF$

$B \Rightarrow b$

$E \Rightarrow aA \mid BSA$

$F \Rightarrow bB \mid b$

Şimdi de $G'_{4.7}$'deki değişkenlerden hangilerinin S'den ulaşılabilen değişkenler olduğunu bulmak gerekir. S'den ulaşılabilen değişkenler kümesi Algoritma 4.6 kullanılarak aşağıdaki gibi bulunur:

$T_U = \{S, B\}$

Yukarıdaki kümede yer almayan A, E ve F değişkenleri yararsız değişkenlerdir. Dilbilgisindeki bu üç değişken ile değişkenlerin yer aldığı kurallar çıkarıldığında a simgesinin de yararsız bir simge olduğu görülür. Bu simge de çıkarıldığında $G_{4.7}$ 'ye eş değer; yararsız simge, değişken ve kural içermeyen aşağıdaki dilbilgisi bulunur:

$G''_{4.7} = <V_N, V_T, P, S>$

$V_N = \{S, B\}$

$V_T = \{b\}$

$P = S \Rightarrow B \mid BS$

$B \Rightarrow b$

4.4.Normal Biçimler

Bağlamdan-bağımsız dilbilgileri için Chomsky Normal Biçimi *(CNF: Chomsky Normal Form)* ve Greibach Normal Biçimi *(GNF: Greibach Normal Form)* olarak adlandırılan iki normal biçim vardır. Bu normal biçimlerin tanımlan ve bağlamdan-bağımsız bir dilbilgisini CNF ve GNF'e dönüştürmek için kullanılan algoritmalar aşağıda verilmektedir.

4.4.1.Chomsky Normal Biçimi

Tanım 4.1. Eğer bağlamdan-bağımsız bir dilbilgisinin yeniden yazma kurallarının tümü

$S \Rightarrow \lambda$

$A \Rightarrow BC$

$A \Rightarrow a$: $A, B, C \in V_N$, $a \in V_T$

biçiminde ise, dilbilgisi Chomsky normal biçimindedir.

Önerme 4.1. Bağlamdan-bağımsız her dil için, bu dili türeten Chomsky normal biçiminde bir dilbilgisi bulunabilir.

Bağlamdan-Bağımsız Dilbilgisinin Chomsky Normal Biçimine Dönüştürülmesi

Bağlamdan-bağımsız bir dilbilgisi verildiğinde, bu dilbilgisine eşdeğer (aynı dili türeten) CNF dilbilgisini bulmak için, dilbilgisinin (A => BC) ve (A => a) biçimindeki yeniden yazma kuralları korunur. Diğer yeniden yazma kuralları atılarak yerlerine aşağıdaki gibi türetilen yeni kurallar konulur.

1. CNF'e uygun olmayan kuralın (K) sağ tarafının ilk simgesi bir değişken ya da bir uç simge olabilir.
    1. Eğer ilk simge bir değişken ise (K):

       $A \Rightarrow BX_2X_3 \ldots X_k$ $A, B \in V_N$ $X_2, X_3, \ldots, X_k \in V$

       Biçimindedir. Bu durumda değişkenler kümesine bir yeni değişken (örneğin D1) eklenir ve (K) yerine aşağıdaki 2 kural konur.
    1. Eğer ilk simge bir uç simge ise (K):

       $A \Rightarrow aX_1X_3 \ldots X_k$ $A \in V_N$ $a \in V_T$ $X_2, X_3, \ldots, X_k \in V$

       Biçimindedir. Bu durumda değişkenler kümesine iki yeni değişken (örneğin C_1 ve D_1) eklenir ve (K) yerine aşağıdaki üç kural konulur.

       $A \Rightarrow C_1D_1$ (K_1)

       $D_1 \Rightarrow X_2X_3 \ldots X_k$ (K_2)

       $C_1 \Rightarrow a$ (K_3)
1. Eğer K'nın yerine konulan kurallardan (K) CNF'e uygun değilse, K için yapılan işlemler K_2 için tekrarlanır ve tüm kurallar CNF'e uygun oluncaya kadar dönüştürme işlemleri sürdürülür.

Örnek 4.8

$G_{4.8} = <V_N, V_T, P, S>$

$V_N = \{S, A, B\}$

$V_T = \{a, b\}$

P: S => aB (1)

S=> bA (2)

A => aS (3)

A => bAA (4)

A => a (5)

B => bS (6)

B => aBB (7)

B => b (8)

Dilbilgisinin 8 yeniden yazma kuralı vardır. Bu kurallardan yalnız ikisi (5 ve 8) CNF'e uygundur. Diğer 6 kuralın yerine aşağıdaki kurallar konulur.

$S \Rightarrow aB$ yerine $S \Rightarrow C_aB$ ve $C_a \Rightarrow a$

$S \Rightarrow bA$ yerine $S \Rightarrow C_bA$ ve $C_b \Rightarrow b$

$A \Rightarrow aS$ yerine $A \Rightarrow C_aS$

$A \Rightarrow bAA$ yerine $A \Rightarrow C_bD_1$ ve $D_1 \Rightarrow AA$

$B \Rightarrow bS$ yerine $B \Rightarrow C_bS$

$B \Rightarrow aBB$ yerine $B \Rightarrow C_aD_2$ ve $D_2 \Rightarrow BB$

Sonuç olarak $G_{4.8'e}$ eş değer Chomsky normal biçimindeki dilbilgisi aşağıdaki gibi oluşur.

$G'_{4.8} = < G_{4.8} = <V_N, V_T, P, S>$

$V_N = \{S, A, B, C_a, C_b, D_1, D_2\}$

$V_T = \{a, b\}$

P: $S \Rightarrow \Rightarrow C_aB \mid C_bA$

$A \Rightarrow C_aS \mid C_bD_1 \mid a$

$B \Rightarrow C_bS \mid C_aD_2 \mid b$

$C_a \Rightarrow a$

$C_b \Rightarrow b$

$D_1 \Rightarrow AA$

$D_2 \Rightarrow BB$

4.4.2.Greibach Normal Biçimi

Tanım4.2.Eğer bağlamdan-bağımsız bir dilbilgisinin yeniden yazma kurallarının tümü

$S \Rightarrow \lambda$

$A \Rightarrow a\alpha \quad : \quad A \in V_N, a \in V_T, \alpha \in V_N^*$

biçiminde ise, dilbilgisi Greibach normal biçimindedir.

Önerme 4.2. Bağlamdan-bağımsız her dil için, bu dili türeten Greibach normal biçiminde bir dilbilgisi bulunabilir.

Bağlamdan-bağımsız bir dilbilgisi GNF'e dönüştürülürken iki özellik (*lemma*) kullanılır. Dönüştürmenin nasıl yapıldığını vermeden önce bu iki özelliğin verilmesi gereklidir.

Lemma 4.1. Bağlamdan-bağımsız bir dilbilgisinin yeniden yazma kurallarından biri:

$$A \Rightarrow \alpha_1 B \alpha_2 \qquad (k_1)$$

olsun. Eğer dilbilgisinin tüm B kuralları

$$B \Rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

İse, dilbilgisinden k1 kuralını çıkarıp yerine

$$A \Rightarrow \alpha_1\beta_1\alpha_2 \mid \alpha_1\beta_2\alpha_2 \mid \ldots \mid \alpha_1\beta_n\alpha_2 \qquad (k_2)$$

Kuralları konulduğunda eş değer (aynı dili türeten) bir dilbilgisi elde edilir.

Lemma 4.2. Eğer bağlamdan-bağımsız bir dilbilgisi doğrudan özyineli (*direct recursive*) yeniden yazma kuralları içeriyorsa, bu dilbilgisine eşdeğer, doğrudan özyineli kural içermeyen bir dilbilgisi bulunabilir. Bunun için doğrudan özyineli kuralların yerine konulacak yeni kurallar aşağıdaki gibi bulunur.

Dilbilgisinin A kurallarının bir kesimi doğrudan özyineli ise, A kuralları iki gruba ayrılır:

$$A => A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_r \quad (g_1)$$

$$A => \beta_1 \mid \beta_2 \mid \ldots \mid \beta_s \quad (g_2)$$

A kurallarından doğrudan özyineli olanların (g_1) yerine aşağıdaki kurallar konulur:

$A => \beta_1 B$ $\quad i = 1, 2, \ldots, s$ $\quad$ B: yeni bir değişken

$B => \alpha_j \quad j = 1, 2, \ldots, r$

$B => \alpha_j B \quad j = 1, 2, \ldots, r$

Bağlamdan-Bağımsız Dilbilgisinin Greibach Normal Biçimine Dönüştürülmesi

Chomsky normal biçiminde (CNF) bağlamdan-bağımsız bir dilbilgisi verildiğinde, aşağıdaki algoritmayı kullanarak bu dilbilgisine eşdeğer (aynı dili türeten) Greibach normal biçiminde (GNF) bir dilbilgisi bulunabilir. Buna göre, GNF'e dönüştürülecek dilbilgisinin önce CNF'e dönüştürülmesi gerekmektedir. GNF'e dönüştürme algoritması 3 adımdan oluşur.

1.Adım

1. Dilbilgisinin söz dizim değişkenleri A_1, A_2, A_3, ..., A_k gibi dizinli değişkenlerle değiştirilir. Bu değişiklik yapılırken S'nin yerine dizin değeri en küçük olan (A_1) değişken konulur.
2. Lemma 4.1 kullanılarak yeniden yazma kurallarının tümü

   $A_i \Rightarrow A_j\gamma \qquad j>=i$

   koşulunu sağlayacak biçime dönüştürülür.

2.Adım

1. Lemma 4.2 kullanılarak doğrudan özyineli kuralların yerine yeni kurallar konulur. Bu adımın sonunda, dizin değeri en büyük (A_k) değişkenle başlayan tüm kurallar GNF'e uygun biçime dönüşmüş olur. Bu adımda dilbilgisine B_1, B_2, … gibi yeni değişkenler de eklenir.

3.Adım

1. Lemma 4.1 kullanılarak A_{k-1}, A_{k-2}, …, A_1, A_2 sırasında tüm A_j kuralları GNF'e uygun biçime dönüştürülür.
2. Lemma 4.1 kullanılarak tüm B_j kuralları (sol tarafında 2.adımda eklenen değişkenlerin yer aldığı kurallar) GNF'e uygun biçime dönüştürülür.

3.adımın sonunda Greibach normal biçimindeki dilbilgisi elde edilmiş olur. Bundan sonra istenirse değişkenler yeniden adlandırılabilir.

Örnek 4.9.

$G_{4.9} =< V_N, V_T, P, S>$

$V_N = \{ A_1, A_2, A_3 \}$

$V_T = \{a, b\}$

P: $A_1 => A_2, A_3$ (1)

$A_2 => A_1, A_2$ (2)

$A_2 => b$ (3)

$A_3 => A_1, A_2$ (4)

$A_3 => a$ (5)

Dilbilgisi CNF biçimindedir. Ancak dilbilgisinin 1, 2 ve 4 numaralı yeniden yazma kurallarını GNF'e uygun değildir. Dönüştürme algoritmasını kullanarak eşdeğer GNF dilbilgisini bulalım.

1. Kural ($A_i \Rightarrow A_j\gamma$, $j>=i$) koşulunun sağlanması.

Bu koşul 1 ve 2 numaralı kurallarda sağlanıyor; 4 numaralı kuralda sağlanmıyor. Lemma 4.1 kullanılarak bu kuralın yerine aşağıdaki kurallar konulur.

Önce $A_3 \Rightarrow A_1A_2$ yerine $A_3 \Rightarrow A_2A_3A_2$ konulur.

Koşul sağlanmadığı için de

$A_3 \Rightarrow A_2A_3A_2$ yerine $A_3 \Rightarrow A_3A_1A_3A_2$

$A_3 \Rightarrow bA_3A_2$ konulur.

Bu adımın sonunda oluşan yeniden yazma kuralları aşağıdaki gibidir:

$A_1 \Rightarrow A_2A_3$

$A_2 \Rightarrow A_3A_1$

$A_2 \Rightarrow b$

$A_3 \Rightarrow A_3A_1A_3A_2$

$A_3 \Rightarrow bA_3A_2$

$A_3 \Rightarrow a$

2. Doğrudan özyineli kuralların yok edilmesi.

Birinci adımın sonunda elde edilen kurallardan yalnız birisi özyinelidir.
Lemma 4.2 kullanılarak:

$A_3 \Rightarrow A_3A_2A_3A_2$ yerine

$A_3 \Rightarrow bA_3A_2B_3$
$A_3 \Rightarrow aB_3$
$B_3 \Rightarrow A_1A_3A_2$
$B_3 \Rightarrow A_1A_3A_2B_3$

B_3: yeni bir değişkendir.

Bu adımın sonunda oluşan yeniden yazma kuralları aşağıdaki gibi$_{dir:}$

$A_1 \Rightarrow A_2A_3$
$A_2 \Rightarrow A_3A_1$
$A_2 \Rightarrow b$
$A_3 \Rightarrow bA_3A_2B_3$
$A_3 \Rightarrow bA_3A_2B_3$
$A_3 \Rightarrow aB_3$
$A_3 \Rightarrow bA_3A_2$
$A_3 \Rightarrow a$
$B_3 \Rightarrow A_1A_3A_2$
$B_3 \Rightarrow A_1A_3A_2B_3$

Tüm kuralların GNF'e uygun biçime getirilmesi.

2.adım sonunda A_3 kurallarının GNF'e uygun biçime geldiği görülür.

3.adımda da Lemma 4.1 kullanılarak A_2, A_1 ve B_3 kuralları GNF'e uygun biçime dönüştürülür. Bunun için de:

$\{A_2 \Rightarrow A_3\ A_1,\ A_2 \Rightarrow b\}$ yerine $\{A_2 \Rightarrow b A_3 A_2 A_1,\ A_2 \Rightarrow a A_1,\ A_2 \Rightarrow b A_3 A_2 B_3 A_1,\ A_2 \Rightarrow a B_3 A_1,\ A_2 \Rightarrow b\}$ konulur.

$A_1 \Rightarrow A_2 A_3$ yerine $\{A_1 \Rightarrow b A_3 A_2 A_1 A_3,\ A_1 \Rightarrow a A_1 A_3,\ A_1 \Rightarrow b A_3 A_2 B_3 A_1 A_3,\ A_1 \Rightarrow a B_3 A_1 A_3,\ A_1 \Rightarrow b A_3\}$ konulur.

$B_3 \Rightarrow A_1 A_3 A_2$ yerine $\{B_3 \Rightarrow b A_3 A_2 A_1 A_3 A_3 A_2,\ B_3 \Rightarrow a A_1 A_3 A_3 A_2,\ B_3 \Rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2,\ B_3 \Rightarrow a B_3 A_1 A_3 A_3 A_2,\ B_3 \Rightarrow b A_3 A_3 A_2\}$ konulur.

$B_3 \Rightarrow A_1 A_3 A_2 B_3$ yerine $\{B_3 \Rightarrow b A_3 A_2 A_1 A_3 A_3 A_2 B_3,\ B_3 \Rightarrow a A_1 A_3 A_3 A_2 B_3,\ B_3 \Rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3,\ B_3 \Rightarrow a B_3 A_1 A_3 A_3 A_2 B_3,\ B_3 \Rightarrow b A_3 A_3 A_2 B_3\}$ konulur.

Sonuç olarak, $G_{4.9}$' a eşdeğer Greibach normal biçimindeki dilbilgisi aşağıdaki gibi oluşur.

$G'_{4.9} = <V_N, V_T, P, S>$

$V_N = \{S, A_1, A_2, A_3, B_3\}$ A_1: Başlangıç değişkeni

$V_T = \{a, b\}$

P: $A_1 \Rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2A_1A_3$

$aB_3A_1A_3 \mid bA_3$

$A_2 \Rightarrow bA_3A_1A_3 \mid aA_1 \mid bA_1A_3A_2B_3A_1 \mid aB_3A_1 \mid b$

$A_3 \Rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$

$B_3 \Rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid$

$bA_3A_2B_3A_1A_3A_3A_2 \mid aB_3A_1A_3A_3A_2 \mid bA_3A_3A_2$

$bA_3A_2A_1A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3$

$bA_3A_2B_3A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_3A_2B_3$

$bA_3A_3A_2B_3$

Örnek 4.10

$G_{4.10} = <V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{a, b\}$

P: $S \Rightarrow aSb \mid ab$

$G_{4.10}$ bağlamdan-bağımsız bir dilbilgisidir. Önce bu dilbilgisini Chomsky normal biçimine dönüştürelim:

P: $S \Rightarrow C_aB$

$B \Rightarrow SC_b$

$S \Rightarrow C_aC_b$

$C_a \Rightarrow a$

Sonra S'nin yerine A1, Ca'nın yerine A2, Cb'nin yerine A3, B'nin yerine de A4 koyarak değişkenleri diinli değişkenler yapalım.

P: $A_1 \Rightarrow A_2A_3$

$A_1 \Rightarrow A_2A_4$

$A_2 \Rightarrow a$

$A_3 \Rightarrow b$

$A_4 \Rightarrow A_1A_3$

Örnek 4.10

$G_{4.10} = <V_N, V_T, P, S>$

$V_N = \{S\}$

$V_T = \{a, b\}$

P: S => aSb | ab

$G_{4.10}$ bağlamdan-bağımsız bir dilbilgisidir. Önce bu dilbilgisini Chomsky normal biçimine dönüştürelim:

P: $S \Rightarrow C_aB$

$B \Rightarrow SC_b$

$S \Rightarrow C_aC_b$

$C_a \Rightarrow a$

Sonra S'nin yerine A1, Ca'nın yerine A2, Cb'nin yerine A3, B'nin yerine de A4 koyarak değişkenleri diinli değişkenler yapalım.

P: $A_1 \Rightarrow A_2A_3$

$A_1 \Rightarrow A_2A_4$

$A_2 \Rightarrow a$

$A_3 \Rightarrow b$

$A_4 \Rightarrow A_1A_3$

Sonra (Ai => Aj, j>=i) koşulunu sağlamayan ($A_4 => A_1A_3$) kuralının yerine yeni kurallar koyalım.

$A_4 => A_1A_3$ yerine $A_4 => aA_3A_3$

$A_4 => aA_4A_3$

Yeniden yazma kurallarının son durumu:

P: $A_1 => A_2A_3$

$A_1 => A_2A_4$

$A_2 => a$

$A_3 => b$

$A_4 => aA_3A_3$

$A_4 => aA_4A_3$

Yeniden yazma kurallarını hiçbiri doğrudan özyineli değildir. Kuralların son dördü de GNF'e uygun biçime gelmiştir. Son olarak A_1 kurallarını GNF'e uygun biçime sokalım. Bunun için

$A_1 => A_2A_3$ yerine $A_1 => aA_3$

$A_1 => A_2A_4$ yerine de $A_1 => aA_4$

Konulur ve Greibach norma biçimindeki dilbilgisi elde edilir.

P: $A_1 => aA_3 \mid aA_4$

$A_2 => a$

$A_3 => b$

$A_4 => aA_3A_3 \mid aA_4A_3$

Elde edilen dilbilgisinde A_2'nin yararsız değişken (A_2 => a)'nın da yararsız kural olduğu görülmektedir. Yararsız değişken ve kuralı atıp, kalan değişkenleri S, B ve C olarak adlandırarak, $G_{4.10}$'a denk, Greibach normal biçimindeki aşağıdaki dilbilgisi bulunur.

$G'_{4.10} = <V_N, V_T, P, S>$

$V_N = \{S, B, C\}$

$V_T = \{a, b\}$

P: S => aB | Ac

B => b

C => aBB | Acb