

Üç Temel Kavram: Diller, Dilbilgisi ve Otomatlar

Alfabe ve Dizgiler

Tanım Dil kavramını tanımlamaya boş olmayan sonlu simgelerden oluşan Σ alfabeti ile başlanır. Bu simgeler teker teker dizilerek dizgileri veya başka bir söyleyişle sözcükleri meydana getirir. Dizgiler de Σ alfabetinin sonlu sayıda simgelerinden meydana gelirler.

Örnek $\Sigma = \{a,b\}$ ise; $ab, aab, abab$ bu Σ alfabeti üzerine oluşturulmuş dizgilerdir.

Tanım Herhangi iki u ve v dizgisinin art arda bağlanması (concatenation) olarak elde edilen uv dizgisi aşağıdaki şekilde simgelenir.

$u = a_1a_2\dots a_n$ ve $v = b_1b_2\dots b_m$ ise, u ile v dizgisinin zincirleme bağlanması(concatenation)

$$uv = a_1a_2\dots a_nb_1b_2\dots b_m$$

Tanım Bir dizginin tersi (reverse) dizginin simgelerinin ters sırada yazılması ile elde edilecektir. Eğer u dizgisi yukarıdaki şekilde verilmişse, bu dizginin tersi u^R dizgisi

$$u^R = \text{rev}(u) = a_n \dots a_2 a_1$$

olarak yazılacaktır. Örneğin 1110 dizgisinin tersi $\text{rev}(1110) = 0111$

Tanım Bir u dizgisinin uzunluğu $|u|$ şeklinde gösterilir ve dizgideki simgelerin sayısını verir. Otomat teorisinin temel dizgisi olan boş dizgi λ simgesi ile tanımlanır ve içinde hiçbir simge bulundurmaz. Ayrıca; λ dizgi hiçbir zaman Σ alfabetinin elemanlarından biri olamaz. Bu dizgi, 0 uzunluğuna sahip olmak üzere $|\lambda| = 0$ ve $\lambda u = u\lambda = u$ bağıntısını da gerçeklemektedir.

$\Sigma = \{0,1\}$ alfabetinin simgelerinin uzunlukları $|0| = 1$ ve $|1| = 1$ olarak yazılır. Benzer şekilde 010 dizgisinin uzunluğu üç iken $|010|_0 = 2$ ve $|010|_1 = 1$ olarak yazılır.

Tanım Eğer " u " herhangi bir dizgi ise, u^n dizgisi, yani u dizgisinin n . kuvveti, özel olarak $u^0 = \lambda$ eşitliğinden başlayarak " u " dizgisinin n defa zincirleme bağlanması ile elde edilir.

Örnek $\Sigma = \{a,b\}$ alfabeti üzerine oluşturulan dizgilerden biri $u = 1110$ ise, bu dizginin dördüncü kuvveti $u^4 = 1110111011101110$ olarak yazılır.

Diller

Σ bir alfabe olmak üzere bu alfabe üzerine oluşturulan Σ^* alfabeti, alfabenin sıfır veya daha fazla sayıdaki simgesinin art arda eklenmesi sonucunda elde edilir. Σ^* alfabeti mutlaka λ dizgisi içerir. Zincirleme bağlama sonucu elde edilmiş yeni kümede λ dizgisi bulunmuyorsa;

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

tanımı yapılır. Σ alfabeti sonlu olmasına rağmen Σ^* ve Σ^+ kümeleri her zaman sonsuzdur. Çünkü bu kümelerin elemanları olan dizgilerin uzunlukları ile ilgili herhangi bir sınır yoktur. Acaba bunlar sayılabilir sonsuz kümeler midir?

Örneğin; $\Sigma = \{0,1\}$ alfabeti için, sıralama şöyle gerçekleşir:

$$\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots$$

Eğer Σ Türkçe alfabe ise, $\Sigma = \{a_1, \dots, a_{29}\}$ sıralanışı $\{a, \dots, z\}$ olarak gerçekleşir. Eşit uzunluktaki dizgilerin sözlük sıralaması, genel amaçlı kullanılan sözlüklerdeki sıralamadır (dictionary order).

Artık hem yapay oluşturulan programlama dillerinin, hem de doğal dillerin tanımı yapılabilir: Σ herhangi bir alfabe olmak üzere, $L \subseteq \Sigma^*$ bağıntısı sağlanıyorsa; L bu alfabe üzerinde bir dil tanımlamaktadır. Kısaca dil en genel şekli ile, bir Σ^* kümesinin altkümesi olarak tanımlanır. \emptyset ve $\{\lambda\}$ bir Σ alfabeti üzerinde dil tanımlarlar. Burada \emptyset dili hiç eleman içermezken, $\{\lambda\}$ dili bir eleman içermektedir. Bu nedenle $\emptyset \neq \{\lambda\}$ yazılır.

Bir L diline ait herhangi bir dizgi ise, bu dilin bir tümcesi olarak adlandırılır.

Örneğin; $L = \{a^n b^n \mid n \geq 0\}$ kümesinin elemanları Σ alfabeti üzerine oluşturulmuştur, $aaabbb$ ve $aaaaabbbbb$ dizgileri bu kümenin elemanlarıdır. Fakat aab dizgisi bu L diline ait değildir. Yukarıda tanımlanan L dili sonsuz elemana sahip olması nedeni ile sonsuz bir dildir.

Tanım Dillerin kümelerle betimlenmeleri nedeni ile, önceki bölümde tanımlanan tüm küme işlemleri, bir başka ifade ile birleşim, kesişim ve fark işlemleri, diller üzerinde işlemler olarak tanımlanmalıdır.

L_1 ve L_2 farklı iki dil olmak üzere, küme işlemleri kullanılarak,

- i. $L_1 \cup L_2 = \{ x \mid x \in L_1 \text{ veya } x \in L_2 \}$
- ii. $L_1 \cap L_2 = \{ x \mid x \in L_1 \text{ ve } x \in L_2 \}$
- iii. $L_1 - L_2 = \{ x \mid x \in L_1 \text{ ve } x \notin L_2 \}$

şeklinde yeni diller oluşturulur. Ayrıca, L dili Σ alfabeti üzerine tanımlanmış olmak üzere, L dilinin tamamlayıcı (complement)

$$\bar{L} = \Sigma^* - L$$

şeklinde tanımlanır.

Tanım L_1 ve L_2 farklı iki dil olmak üzere, bu dillerin zincirleme (art arda) bağlanması ile meydana gelen yeni dil

- $L_1 L_2 = \{ xy \mid x \in L_1 \text{ ve } y \in L_2 \}$ olarak tanımlanır ve her L dili
- i. $L\{\lambda\} = \{\lambda\}L = L$
 - ii. $L\emptyset = \emptyset L = \emptyset$ bağıntılarını gerçekler.

Tanım L bir Σ alfabeti üzerine oluşturulmuş herhangi bir dil olsun. L dilinin Kleene yıldızı ya da Kleene işleci veya başka bir ifade ile Kleene kaplaması ise L^* şeklinde gösterilir ve

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

formülü ile sıfır veya daha fazla sayıdaki L dilinin zincirleme bağlanması ile elde edilir. L dilinin pozitif kaplaması L^+ ise, önceki bölümde Σ alfabeti için yapılan Σ^+

tanımına benzer şekilde $L^+ = \bigcup_{i=1}^{\infty} L^i$ formülü ile, bir veya daha fazla L dilinin art arda

bağlanması ile oluşturulur. Her L dili aşağıdaki iki bağıntıyı mutlaka sağlamalıdır:

Dilbilgisi

İnsanoğlunun herhangi bir dil işlemcisi olduğu kabul edilsin. “Köpek şapkanın içinde uyuyor” şeklindeki bir tümce sözdizimsel olarak doğru olduğu için işlemci yani insanoğlu tarafından tanınır. Burada böyle bir eylemin gerçekleşip gerçekleşmediği önemli değildir. Oysa “içinde köpek şapkanın uyuyor” tümcesi kolay anlaşılır bir söz olmamasına rağmen, yeniden düzenlenerek tümce yapıları için kabul edilmiş genel kurallara uygun olup olmayacağı işlemci tarafından belirlenebilir. Bu durumda kişi, geçerli dizgileri kabul eden bir dil tanımlayıcısı (language recognizer), yani bir aygıt gibi davranmaktadır.

Herhangi bir doğal dilde sözcükler çeşitli şekillerde birleştirilir. Bu işlemi gerçekleştiren dilbilgisi, sözcüklerinin birleştirilmesi ile oluşturulan tümcenin geçerli olup olmadığını araştırır. Burada önemli olan elde edilen anlam değil sözcüklerin nasıl birleştirildiği, yani sözdizimidir. Programlama dillerinin sözdizimi ise, genellikle dilbilgilerine bağlı belirtim araçlarının kullanılması ile betimlenir. Biçimsel bir dilin tümceleri de herhangi bir dilbilgisine uygun olarak tanımlanır.

Dilbilgisi, programlama dilleri uygulamalarında sorulabilecek şu soruları cevaplamaya yardımcı olur:

- i. Sözcüklerin herhangi bir birleşiminin bir biçimsel dilde geçerli bir tümce oluşturup oluşturmadığı nasıl belirlenir?
- ii. Bir biçimsel dildeki geçerli tümceler nasıl oluşturulabilir?

Yukarıdaki soruların cevabı, ilki için dil tanıma (language recognition), ikinci soru için ise dil üretimi (language generation) olacaktır.

Bir işlemci olarak düşünüldüğünde insanın dil üretici olarak davranması şöyle gerçekleşir: “Köpek şapkanın içinde uyuyor” tümcesini oluşturmak ve aynı zamanda söylemek (speech) için, biçimsel dil üretici görevini gören aygıt bir başlangıç imi ile işlemlemeye başlar. Üretcin çalışma süreci başlangıç iminin kullanımı ile tam olarak belirlenmemiş olmasına rağmen, tanımlanmış olan kurallar kümesi ile sınırlıdır. En sonunda bu süreç durur (halt) ve aygıt tamamlanmış dizgiyi çıktı olarak verir. Aygıt tarafından tanımlanan dil, oluşturabileceği tüm dizgilerin kümesidir.

Noam Chomsky'nin dil çalışmalarına katkısı evrensel dilbilgisi (universal grammar) olarak bilinir ve şöyle açıklanır. İnsan beyni ana dilini oluşturmak için sınırlı sayıda kural içerir. Aslında her dilin kendisine ait genel yapısal kuralları vardır. Bu kurallar kümesi evrensel dilbilgisi olarak adlandırılır; doğuştan var olan ve herkes tarafından paylaşılan dilbilgisi ilkeleri evrensel dilbilgisini tanımlar. Kısaca evrensel dilbilgisi belirli dilleri tanımlamaktansa dil edinimini genel olarak açıklamayı hedefler ve çocuğun gelişimindeki dil kazanımını açıklamak üzere tasarlanan bir dizi kural önerir.

Ne dil tanıyıcının ne de üreticinin herhangi bir doğal dilde oluşturulması kolay değildir. Doğal dillerin çok büyük altkümeleri için bu tür aygıtların tasarımı uzun yıllardır tartışılmakta ve üzerinde çalışılmaktadır. Burada önemli olan, biçimsel üreticiler teorisi olarak hesaplama teorisinin başlıca kavramlarından olan düzgün diller gibi yapay dillerin oluşturulmasıdır.

Dilbilgisi dildeki cümleleri sıralayan bir aygıt olarak düşünülebilir. Genel bir dilbilgisi dört farklı tipte incelenebilir:

- i. Kuralcı (prescriptive) dilbilgisi dile ait geçerli normları öngörür.
- ii. Tanımlayıcı (descriptive) dilbilgisi keyfi kuralların zorlanmasından ziyade, dilin gerçek kullanımının betimlenmesini hedefler.
- iii. Biçimsel (formal) dilbilgisi kesin olarak tanımlanmış dilbilgisi kurallarıdır.
- iv. Üretken (generative) dilbilgisi doğal dil ifadelerini üretebilen biçimsel bir dilbilgisidir.

Formal dillerde yaygın olarak kullanılan iki dilbilgisi sınıflandırması üretken dilbilgisi ile analitik, yani çözümleyici dilbilgisidir. Üretken dilbilgisinde dile ait dizgilerin üretileceği bir algoritma kullanılırken, analitik bir dilbilgisi giriş dizgisinin verilen dile ait olup olmadığını 0 ya da 1 şeklinde mantıksal şekilde cevaplayacak bir dizi kural ile tanımlanır. Bir başka bakış açısından değerlendirilecek olursa, üretken dilbilgisi bir dilin nasıl yazılacağını betimlerken, analitik dilbilgisi bu dilin nasıl okunacağını bir çözümleyici aracılığı ile betimler.

Dilbilgisinin önemi “bir dizginin herhangi bir dile ait olup olmadığı nasıl belirlenir?” sorusunu cevaplayarak vurgulanabilir.

Dilbilgisi bir dizi türetme kuralı ile bir dilin alfabesinden elde edilebilen dizgilerin sözdizimsel olarak dile ait olup olmadığını açıklar. Dilbilgisi kurallarına göre oluşturulan dile ait dizgiler, dilbilgisinin başlangıç simgesi ile başlayıp türetme kurallarının keyfi bir sıralanışla uygulanması ile elde edilir. Örneğin Σ alfabesi a ve b simgelerinden oluşmak üzere S başlangıç simgesi olsun ve dilbilgisine ait türetme kuralları

$$S \rightarrow aSb$$

$$S \rightarrow ba$$

şeklinde tanımlansın. \rightarrow sembolü sol taraftaki S simgesinin sağ taraftaki semboller ile yer değiştireceğini ifade eder. S simgesi Σ alfabesinin elemanlarından biri, yani alfabenin sembollerinden biri olamaz ve nonterminal sembolü olarak adlandırılır.

Dile ait herhangi bir sözcüğü türetmek için başlangıç nonterminali olan S ile başlanır ve yukarıdaki türetme kurallarından birincisi seçildiğinde aSb dizgisi elde edilir. Daha sonra yine birinci kural seçildiğinde S nonterminali aSb türetme kuralı ile yer değiştirir ve aSb \Rightarrow aaSbb dizgisine ulaşılır. Türetmeye S nonterminali yerine birinci kuralın uygulanması ile devam edilir. İkinci kural uygulandığında artık S nonterminali kalkmıştır ve sadece a ve b simgelerinden oluşan aranan dizgi elde edilmiştir.

Chomsky'nin üretici dönüşümsel (generative transformational) dilbilgisi tanımında her bir türetme adımı \Rightarrow sembolü ile ilerler. Genel ifadesi $\alpha, \beta \in (N \cup \Sigma)^*$ olmak üzere $\alpha \Rightarrow \beta$ şeklindedir. β dizgisi α dizgisinden elde edilmektedir ve dilbilgisi kuralının sol tarafındaki bazı dizgiler sağ tarafındaki bazı dizgilere dönüşmektedir. Eğer $\alpha = \alpha_1 A \alpha_2$ ve $\beta = \beta_1 \gamma \beta_2$ ise bu tek türetme adımında $A \rightarrow \gamma$ dilbilgisi kuralı kullanılmıştır. Bu işlem " α dizgisi β dizgisini türetir" ya da " β dizgisi α dizgisinden tek adımda türemiştir" şeklinde yorumlanır. Genel olarak $\alpha \Rightarrow^* \beta$ ifadesi ile α dizgisinin β dizgisini sıfır veya daha fazla sayıda adım ile türettiği anlaşılır.

Eğer $\alpha = \beta$ ise öyle bir $k \geq 1$ tamsayısı ve $\alpha_0 \alpha_1 \dots \alpha_k$ vardır ki $\alpha_0 = \alpha$ ve $\alpha_k = \beta$ sağlanmalıdır.

Sonuç olarak; $S \rightarrow aSb$

$S \rightarrow ba$

dilbilgisi kuralları ile adım adım gerçekleştirilen türetme sonucu

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow^* \dots \Rightarrow aa \dots ababb \dots b$ $|a|=|b|$

olarak gerçekleşir ve sadece Σ alfabesinin simgelerinden oluşan bir dizgiye ulaşılmıştır.

Bu dilbilgisi kurallarına göre oluşturulan dizgilere ait L dili de

$\{ba, abab, aababb, aaababbb, \dots\}$ kümesi olarak elde edilir.

Tanım Dilbilgileri dil üreticiler ya da diller için dizgi tanıyıcılar olarak kullanılırlar.

Dilbilgisi $G = (N, \Sigma, S, P)$ şeklinde 4-lülerle tanımlanır.

Bu tanımlamada: Σ dile ait sözcükler, yani terminal (giriş) simgelerinin sonlu kümesini veren alfabe, N giriş alfabesine ait sembollerin genelleştirilmesi amacı ile kullanılan terminal olmayan simgelerin sonlu kümesi, S başlangıç nonterminali, P $\alpha, \beta \in (N \cup \Sigma)^*$ olmak üzere $\alpha \rightarrow \beta$ biçimindeki kuralların sonlu kümesi

P kümesinin elemanlarından en az bir tanesi için $\alpha = S$ olmalıdır. Bu da $|\alpha| \leq |\beta|$ koşulunun sağlanması demektir. Ayrıca $N \cup \Sigma \neq \emptyset$ koşulu mutlaka sağlanmalıdır.

$\alpha \rightarrow \beta$ türetme kuralları için $(N \cup \Sigma)^+ \rightarrow (N \cup \Sigma)^*$ ifadesi kullanıldığında türetme kurallarının sol tarafının hiçbir zaman boş olamayacağı da gösterilmiş olur. Dilbilgisinin tipini türetme kurallarının sol tarafı, sağ tarafı veya her iki taraftaki değişiklikler belirler.

Bir G dilbilgisinin üretken bir dilbilgisi olması sadece ve sadece dile ait dizgileri türetmesi demektir. G dilbilgisine ait dil $L(G)$ şeklinde gösterilir ve bu dilbilgisi tarafından Σ alfabesi üzerine oluşturulan tüm dizgilere ait dil

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ şeklinde ifade edilir.

$\Sigma = \{a, b, c\}$, $N = \{S, B\}$ ve S başlangıç nonterminali olmak üzere G dilbilgisine ait türetme kuralları

$S \rightarrow aBSc \mid abc$

$Ba \rightarrow aB$

$Bb \rightarrow bb$

ile veriliyor. $L(G)$ kümesine ait 3 farklı dizgi şöyle elde edilecektir:

1. Dizgi : $S \Rightarrow abc$

2. Dizgi: $S \Rightarrow aBSc \Rightarrow aBabcc \Rightarrow aaBbcc \Rightarrow aabbcc$

3. Dizgi: $S \Rightarrow aBSc \Rightarrow aBaBSc \Rightarrow aBaBabccc \Rightarrow aaBBabccc \Rightarrow aaBaBbcc \Rightarrow aaaBBbcc \Rightarrow aaaBbbccc \Rightarrow aaabbccc$

Chomsky Sıradüzeni

Noam Chomsky tarafından 1956 yılında Chomsky sıradüzeni olarak adlandırılan dilbilgisi sınıflandırmasında düzgün dilbilgisi (regular grammar) 3. tip, bağlamdan bağımsız dilbilgisi (context-free grammar) 2. tip dilbilgisi olarak adlandırılır. Bu şekilde devam edildiğinde kuralların biraz daha serbest değişebilmesi ile sıradüzen içindeki bağlama duyarlı dilbilgisi (context-sensitive grammar) 1. tip, sınırsız dilbilgisi de (unrestricted grammar) 0. tip dilbilgisi olarak adlandırılır

Bu sıradüzen içerisinde farklı dilbilgisi tipleri arasında

Tip 3 \subset Tip 2 \subset Tip 1 \subset Tip 0 bağıntısı görülebilir.



Backus-Naur Form

Programlama dillerinin sözdizimi çoğunlukla dilbilgisine bağlı araçların betimlenmesi ile tanımlanır. Programlama dillerini betimleme araçlarından üçü, Backus-Naur form (BNF), genişletilmiş Backus-Naur form ve sözdizimi çizgeleridir. Dilbilgileri en genel şekli ile Backus-Naur-Form (BNF) simgeleminde yazılırlar. Bu simgelem John Backus tarafından 1959 yılında tanımlanmış Peter Naur tarafından da ALGOL 60 programlama dilinin belirtiminde kullanılmak üzere yeniden değerlendirilmiştir

Backus-Naur formu iki tür simge içerir: terminal sembolleri ve terminal olmayan semboller. Terminal sembolleri sözcükleri simgelerken, terminal olmayan semboller, bir başka ifade ile nonterminaler ifadeler gibi sözdizimsel yapıları betimler. Backus-Naur formunun temeli sonlu sayıda ürünlerin (productions) kümesidir. Her bir ürün

$$A \rightarrow x_1 \mid x_2 \mid \dots \mid x_n$$

biçimindedir. Bir dilbilgisi gösterimi olan \rightarrow yerine, Backus-Naur formda ::= sembolü “simgeleminde içerebilir” anlamında kullanılır. Aynı zamanda tüm nonterminal sembolleri, yani terminal olmayan elemanlar, $\langle \rangle$ imleri arasına yazılır. Örneğin, $A \rightarrow Aa$, $A \rightarrow a$, $A \rightarrow AB$ kuralları \mid barının “veya alternatif olarak” anlamında kullanılması ile, aşağıdaki şekilde birleştirilebilir.

$$\langle A \rangle ::= \langle A \rangle a \mid a \mid \langle A \rangle \langle B \rangle$$

Örnek Bilgisayar bilimlerinin temel uygulamalarından biri sözdizimsel çözümlmeye örnek oluşturan ve bir programlama diline ait değişken tanımlamalarının genelleştirildiği kuralların Backus-Naur formda aşağıdaki şekilde yazılmasıdır.

$$\begin{aligned} \langle \text{program} \rangle &::= \text{program} \langle \text{tanıtıcı} \rangle (\langle \text{tanıtıcı listesi} \rangle); \langle \text{blok} \rangle \\ \langle \text{tanıtıcı listesi} \rangle &::= \langle \text{tanıtıcı} \rangle \{ , \langle \text{tanıtıcı} \rangle \} \end{aligned}$$

Bu örnekte kullanılan $\{ \}$ gösterimi bir Backus-Naur form notasyonudur ve içerideki ifadenin sıfır veya daha fazla sayıda tekrarlanacağını ifade eder. Bu gösterim aynı zamanda özyineli tanımını vurgulamaktadır. $\langle \text{tanıtıcı listesi} \rangle$ isimli değişkenin yukarıdaki türetme kuralı $\{ \}$ gösterimi kullanılmadan

$$\langle \text{tanıtıcı listesi} \rangle ::= \langle \text{tanıtıcı} \rangle , \langle \text{tanıtıcı listesi} \rangle \mid \langle \text{tanıtıcı} \rangle$$

olarak ta yazılabilir. tanıtıcı listesi isimli değişken, imi ile ayrılmış bir veya daha fazla tanıtıcı değişkeninin dizildiğini vurgular. Pascal programlama dilinin sözdiziminin çoğu bu şekilde oluşturulmuştur.

Örnek “Kırmızı başlıklı kız kırmızı kitabın sayfasını sabahleyin yırttı” tümcesini çözümleyecek dilbilgisinin Backus-Naur biçiminde nasıl yazılabilir?

<tümce> ::= <ad öbeği> <eylem öbeği>
<ad öbeği> ::= <sıfat> <ad öbeği>
<ad öbeği> ::= <ad (eki)>
<eylem öbeği> ::= <zarf> <eylem öbeği>
<eylem öbeği> ::= <eylem (ekler)>
<sıfat> ::= kırmızı / başlıklı
<ad(eki)> ::= kız / kitabın / sayfasını
<zarf> ::= sabahleyin
<eylem> ::= yırttı

Örnek İşaretili tamsayıların ondalıklı gösterimde if-then kuralı ile Backus-Naur biçimi nasıl yazılabilir?

<işaretili tamsayı> ::= <işaret> <tamsayı>
<işaret> ::= + / -
<tamsayı> ::= <rakam> / <rakam> <tamsayı>
<rakam> ::= 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9

Otomatlar

Otomat sözcüğü hesaplamaların otomatik modelini tanımlar. Bu modelde işlemler asansörler, fırınlar, stereo sistemlerdeki gibi pek çok farklı kontrol biriminin gerçekleştirdiği etkileşimlerde olduğu gibi durumlarla gerçekleştirilir. Bu sistemlerin tümü içerdiği sabit sayıdaki durumdan birinde olmalıdır. Örneğin; asansör kapısı açık veya kapalı olarak katlardan birinde olabilir veya katlar arasında hareket halinde olabilir, asansörün dışından veya içinden belli bir yöne (yukarı veya aşağıya) bir çağrı yapılmış olabilir. Sistemin o anki durumu daha sonra ne yapılacağı hakkında fikir verecektir. Asansör örneğindeki basit sistemde olduğu gibi, insan yapısı her makine bir sonlu durum sistemidir. Durumların sayısı çok arttıkça, sonlu durum modeli uygun şekilde durdurulur. Buradan bilgisayarların da birer sonlu durum sistemi olduğu sonucu çıkarılacaktır. Her bitinde 1 veya 0 depolayan bilgisayarların bellek ve yazmaçları (registers) sonlu durum sistemi ile tasarlandıklarında, çok fazla sayıda durum içereceklerdir. Örneğin; 32 mega bayt belleğe sahip bir makine $10^{81,000,000}$ farklı durumla tasarlanabilir. Böyle bir makinenin bir sonlu durum makinesi oluşturduğunu söylemek imkansızdır. Ama yine de, sonlu durum modellerinin belli başlı uygulama alanları olarak şunlar verilebilir:

- i. Aritmetik ve mantıksal birimler, arabellek (buffers) veya I/O kotarıcıları (handlers) gibi mantıksal devre tasarımında,
- ii. lex, grek, awk gibi Unix araçları olarak bazı programlara destek veren programlar (utilities),
- iii. Düzenleyici programlarının örüntü eşleme araçları şeklinde

tasarlanabilirler. Çünkü bu örneklerin hemen hemen tümü sayıları çok fazla olmayan duruma sahiptir.

Otomatlar sayısal (dijital) bilgisayarların soyut birer modelidir ve her otomat kendisine özel bazı temel özellikler içerir. En temel özellikleri de girişleri okuyabilen mekanizmalar olmalarıdır. Giriş, bir alfabe üzerine oluşturulur ve sadece okuma işlemi yapabilen bir giriş dosyasına yazılır. Bu giriş dosyasında herhangi bir değişiklik yapılması mümkün değildir. Giriş mekanizması bu dosyayı soldan sağa doğru her seferde tek bir sembol olarak okur. Giriş mekanizması ayrıca giriş dizgisinin sonunu da kontrol eder ve bu işlemi bir dosya sonu işlemi ile gerçekleştirir.

Hesaplanabilir modeller üç sınıfta gruplandırılabilir:

- i Sonlu otomatlar veya sonlu durum makineleri
- ii Son giren ilk çıkar otomatlar
- iii Turing makineleri

Modellerin bu şekildeki kavramsallaştırılması hesaplamanın pragmatik olarak yönlendirilmiş bir yaklaşımına dayanmaktadır. Aslında bu üç temel model bilgisayarların tümüyle ortak yazılım ve donanım bileşenleri üzerindeki çalışmalar sonunda tasarlanmaktadır. Bu modellerin önemi ise, bilgisayar biliminin bu bilgisayar bileşenlerini soyutlamasını kolaylaştırmasıdır.

Sonlu Otomatlar

Sonlu otomatlar çıktı içeren sonlu durum makineleri ve çıktı içermeyen sonlu durum makineleri olarak iki sınıfa ayrılırlar. Pek çok otomatik makine bilgisayar bileşenleri içermesinden dolayı, bir sonlu durum makinesi ile tasarlanabilir. Sonlu otomatlar, özellikle sınırlı bellek miktarı gerektiren bilgisayarlar için çok uygundur. Böyle düşük bellek ile bilgisayarların yapabileceklerinin sınırlı olacağı düşünülmesine rağmen, bu tür bilgisayarlar ile gündelik hayatta oldukça sık karşılaşmaktadır. Örnek olarak para makineleri verilebilir. Bu makineler para veya yiyecek/içecek verdikleri için çıktı içeren sonlu durum otomatları olarak adlandırılırlar. Ayrıca; çıktı içeren sonlu durum makinelerine, giriş dizgilerini belli miktarda geciktiren veya tamsayıları toplayan soyut makineler örnek olarak verilebilir. Bunlar sayısal devrelerin tasarımında yararlanılan sonlu otomatlar şeklinde genelleştirilebilir. Bu tür çıktı içeren sonlu durum makineleri, her bir devrimde çıktı alfabesinden bir simge alacaktır. Kısaca; dönüştürücü (transducer) adı verilen bu otomatlar, giriş alfabesindeki bir simgeyi, çıkış alfabesindeki bir simgeye dönüştürmektedir.

Çıktı içermeyen sonlu durum otomatları da belirlenimci sonlu durum otomatları ve belirlenimci olmayan sonlu durum otomatlar şeklinde iki temel grupta incelenir. Belirlenimci bir otomat her devrimin yürürlükteki düzenleştirmesi tarafından tek olarak belirlendiği tiptir. İç durum, giriş ve çalışma belleğinin içerikleri biliniyorsa, otomatın sonraki davranışı tam olarak tahmin edilebilir. Belirlenimci olmayan sonlu durum otomatı her noktada pek çok devrimine sahip olabilir; bu nedenle de bir dizi olası eylemin içinden tahmin yapmak gerekecektir.

Çıktı içermeyen sonlu durum otomatına örnek olarak herhangi bir HTML dokümanında yazar – eser çifti listesinin oluşturulması olsun. Listenin taranması sonunda böyle bir bilgiye ulaşılabiliyorsa, otomat kabul edilecektir. Bunun için isim ve eser ile eşleşecek dizgiler araştırılarak bunlar çift olarak bir tabloya eklenecektir. Buna ait HTML kodu

```
<OL> ----- NUMBERED | ORDERED LIST
<UL> ---- UNNUMBERED | UNORDERED LIST
şeklinde tanımlanır.
<OL>
```

```
<L1> SAYGUN BY HALAY </L1>
```

```
<L1> UN BY YURDUM</L1>
```

yukarıdaki HTML koduna bir örnektir.

İkili sayı sisteminde verilmiş bir w dizgisinin 01 örneğini içerip içermediğini araştırılan problem de çıktı içermeyen bir sonlu durum otomatı ile çözümlenebilir. Burada

$\Sigma = \{0,1\}$ alfabeti üzerine oluşturulan

$L = \{w \in \Sigma^* \mid w \text{ dizgisi } 01 \text{ alt dizgisi içerir}\} = \{x,y \in \Sigma^* \mid x01y\}$

dili 00101 , 000111 dizgilerini içerirken; 111000 dizgisi L diline ait değildir.

Son Giren İlk Çıkar Otomatlar

Sonlu durum otomatlarının gücünü arttırmak üzere, bunlara sınırsız bellek eklemesi yapılabilir. Durumların sayısını arttırarak genişleyecek olan bellek, makinelerin daha karmaşık dilleri tanımalarına olanak sağlar. Fakat incelenen dilin, Bölüm 6'da ayrıntılı olarak anlatılacağı gibi, düzgün bir dil olup olmamasına göre, yeni otomat ta problemin çözümü için yeterli olacak veya yetersiz kalacaktır. Aslında sonlu sayıda durum ile düzgün olmayan bir dili simgelemek mümkün değildir. Sonsuz sayıda durum ise, makinenin belirtilmelerini sonlu olarak tanımlayamaz. Bağımsız olarak ilave edilecek olan bellek, son giren ilk çıkar ilkesine göre çalışan bir ters bellektir ve sonlu durum otomatlarının tanımadığı düzgün olmayan dilleri de tanıyabilir. Son giren ilk çıkar otomatlar ise bu tür dillerle ilişkilendirilerek, bu dillere ait dizgileri tanıyan makineler oluşturur.

Bu yeni otomatların işleyişi sonlu durum otomatlarında olduğu gibi devinimler şeklinde gerçekleşir. Fakat son giren ilk çıkar otomatlarda bir devinim sadece mevcut duruma ve mevcut giriş simgesine değil, aynı zamanda ters bellek ya da başka bir deyişle yığıtın en üstündeki sembole de bağlıdır; her adımda okunan giriş simgesi yığıtın en üstüne yerleşerek kontrol biriminin durumu değişir veya aynı kalır. Aynı zamanda da okuyucu kafa ya yerinde kalır ya da bir birim sağa kayar. Bir son giren ilk çıkar otomatın herhangi bir giriş dizgesini kabul edebilmesi için, kafanın en sağa gelerek (giriş dosyasında okunacak simge kalmamak üzere) makinenin final durumuna gelmiş olması ve ters belleğin de boşalmış olması gereklidir.

Turing Makineleri

Sonlu durum otomatları ile son giren ilk çıkar otomatlar arasındaki temel fark, son giren ilk çıkar otomatların geçici bellek özelliğinin olmasıdır. Bellek içermeyen soyut makine bir sonlu durum otomatıdır; eğer bellek bir yığıt ise, sonlu durum otomatından daha güçlü olan son giren ilk çıkar otomatı tasarlanmaktadır. Bu gözlemden yola çıkarak her ikisinden de daha etkin dil aileleri için, daha esnek belleğe sahip bir otomat tasarlanabilir. Turing makineleri sonlu durum otomatları ve son giren ilk çıkar otomatların tanıyamadığı karmaşık diller için geliştirilmiştir. Bu aşamada şöyle bir soru da akla gelmektedir: Eğer tasarlanan herhangi bir otomatta iki yığıt, üç yığıt, tek kuyruk gibi farklı bellek aygıtları kullanılırsa, her yeni bellek yeni bir otomat ve buna bağlı olarak yeni bir dil tanımı mı gerektirecektir? Aslında; en güçlü otomatın nasıl tanımlanabileceğine, buna bağlı olarak ta hesaplama sınırlarının neler olduğuna karar verildiğinde otomat kavramının da ne ölçüde genişletilebileceğine karar verilebilir. Turing makinesi de böyle bir yaklaşım ile temel bir kavram olarak ortaya çıkmakta ve mekanik ya da algoritmik hesaplama fikrinin kesin bir tanımını vermektedir.

Bilgisayarların yapabildiklerini ve yapamadıklarını ifade etmeyi hedefleyen Turing Makinesi temel bileşen olarak hem sonlu-durum makinesi hem de ilave bellek gerektirir. Son giren ilk çıkar otomatın bellek olarak yığıt kullanmasına rağmen, Turing Makinesi her iki yönde sonsuz uzunlukta olduğu düşünülen bir manyetik bant (teyp) kullanır. Bant her biri tek simge kullanabilen karelerden oluşmaktadır. Bant kafası, bir başka ifade ile okuma-yazma kafası, banttan bir simge okuyabilir; banda bir simge yazabilir ve bandı her iki yönde bir kare hareket ettirebilir. Şimdiye kadar incelenen otomatların aksine Turing makinesi giriş dizgesini okumaz. Giriş ve çıkış sembolü gerektiğinde manyetik bant kullanılacaktır.

Alıcılar otomat üzerinde işleme sonucunda kabul veya red olarak sadece ikili çıktı verdikleri, dönüştürücülerin ise daha karmaşık sonuçlar oluşturabildikleri belirtilmişti. Sonlu durum otomatı ve son giren ilk çıkar otomatlar birer alıcıdır. Turing Makinesi de giriş dizgesini kabul veya reddeder. Ayrıca, makine işlemlerini bitirdiğinde şerit üzerinde kalan sonuçlar hesaplamanın çıktısı olarak düşünülebilir. Bu nedenle de Turing makinesi bir dönüştürücü özelliğine de sahiptir.