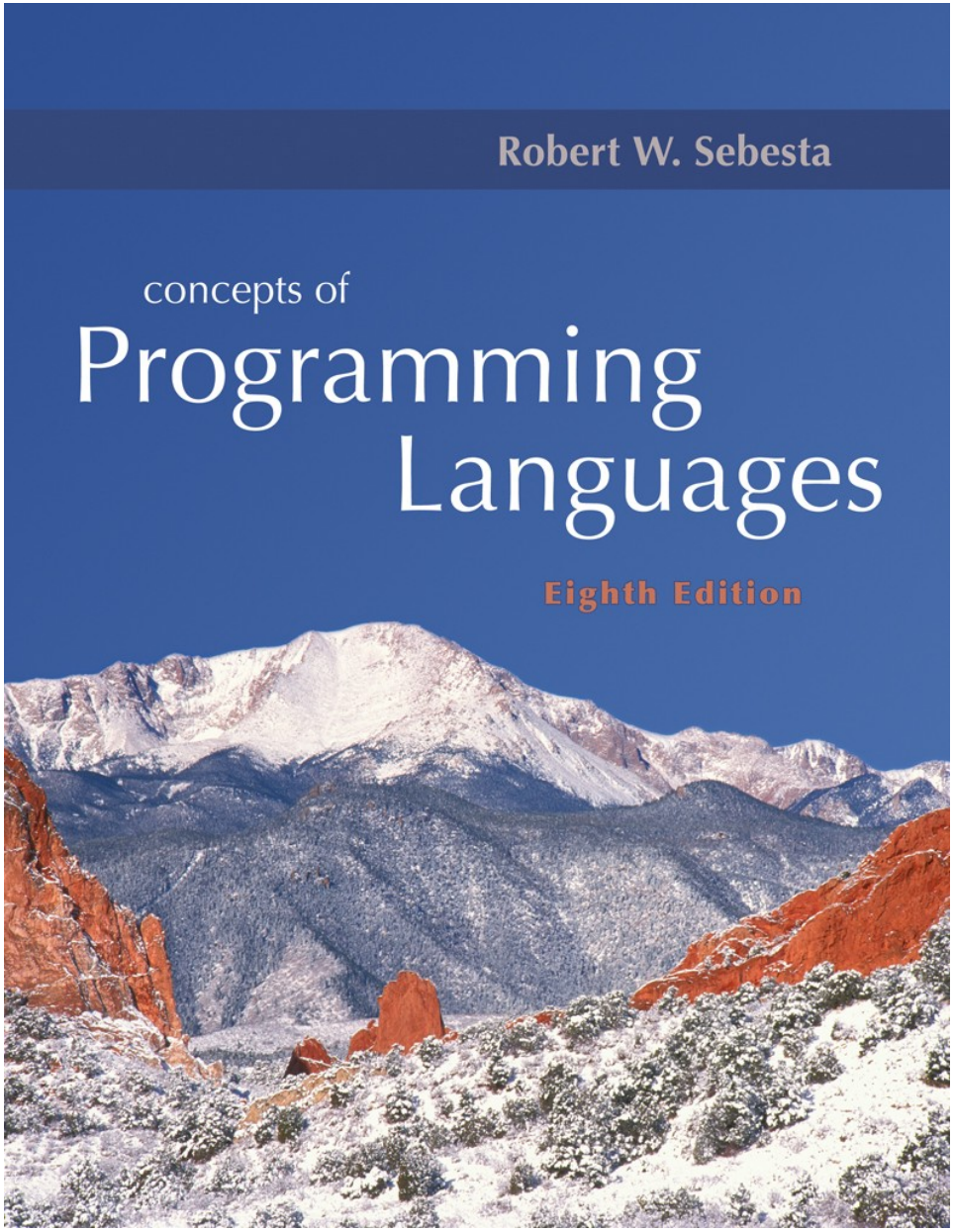
Bölüm 4

Sözcüksel ve sözdizimsel analiz



ISBN 0-321-49362-1

Bölüm 4 Konular

- giriş
- Sözcüksel Analiz
- Ayrıştırma(parsing) problemi
- Özyinelemeli aşağıya iniş ayrıştırma(Recursive-Descent Parsing)
- Aşağıdan-yukarıya ayrıştırma(Bottom-Up Parsing)

Giriş

- Dil gerçekleştirim sistemleri kaynak kodu gerçekleştirime bakmaksızın analiz etmelidir.
- Neredeyse tüm söz dizimsel analiz kaynak dilin biçimsel tanımlanması temellidir (BNF).

Söz dizimsel Analiz

- Söz dizimsel analiz iki parçadan meydana gelir:
  - Düşük seviye kısım sözcüksel analizci(*lexical analyzer)bir sonlu otomatadır.*
  - Yüksek seviye kısım; bir sözdizim analizcisi(*syntax analyzer)* ,veya ayrıştırıcı(parser) serbest içerik temelli bir aşağı-itme otomatı temellidir.(push-down automaton)

Sözdizimi tanımlamak için BNF kullanma

- Temiz ve kısa bir söz dizim tanımı sağlar
- Ayrıştırıcılar(parsers) direkt olarak BNF temelli olabilir.
- BNF temelli olan ayrıştırıcılar kolay bakımı yapılıp çalıştırılabilirler.

Sözcüksel(lexical) ve sözdizimsel(syntax) analizi niye ayırıyoruz?

- Basitlik– lsözcüksel analizde daha az karmaşık olan yaklaşımlar vardır. Ayırdığımızda ayrıştırcı(parser) daha kolay gerçekleştirilebilir.
- Verimlilik– Ayırarak sözcüksel analizci(lexical analyzer) üzerinde optimizasyon yapabiliriz.
- *Taşınabilirlik* – Sözcüksel analizcinin parçaları taşınamaz ama ayrıştırıcının taşınabilir.

Sözcüksel Analiz(Lexical Analysis)

- Sözcüksel analizci verilen karakter dizisi(string) içinde belirli şablona uyan alt stringleri bulur (pattern matcher)
- Sözcüksel analizci ayrıştırıcının ön kısmıdır.
- Kaynak program içindeki lexeme ları belirler.
  - Bulunan lexeme lar kategorilerini belirten token larla eşleştirilir.
  - `toplam` bir lexeme; onun token ı `IDENT`

Sözcüksel Analiz

- Sözcüksel analizci ayrıştırıcının çağırdığı bir fonksiyondur.Ayrıştırıcı bir sonraki lexeme ve token a ihtiyaç duydugunda çağırır.
- Sözcüksel analiz gerçekleştiriminde üç yaklaşım vardır:
  - Token ların biçimsel tanımlarını yap ve yardımcı programlar yardımıyla tablo tabanlı bir sözcüksel analizci oluştur.
  - Bir durum diyagramı(state diagram) tanımla. Bu diyagram tokenları ve aralarındaki geçişleri ifade eder.
  - Bir durum diyagramı tanımlama ve el ile oluşturulan bir tablo tanımla. Bu tablo token lar ve durumlar arasındaki ilişkileri tanımlar.

Durum diyagramı Tasarımı(State Diagram)

- Bir doğal durum diyagramı kaynak koddaki her karakter için bir geçiş(transition) tanımlar. Bu şekildeki diyagram oldukça geniş olur.

Sözcüksel Analiz

- Birçok durumda, geçişler birleştirilerek oluşturulacak olan durum diyagramı küçültülür.
  - Bir identifier tanırken tüm büyük ve küçük harfler eşdeğerdir.
    - Tüm karakterler için karakter(character) sınıfını kullan
  - Bir tamsayı değişmezi(integer literal) tanırken tüm rakamlar(digit) eşdeğer digit sınıfı kabul edilir.

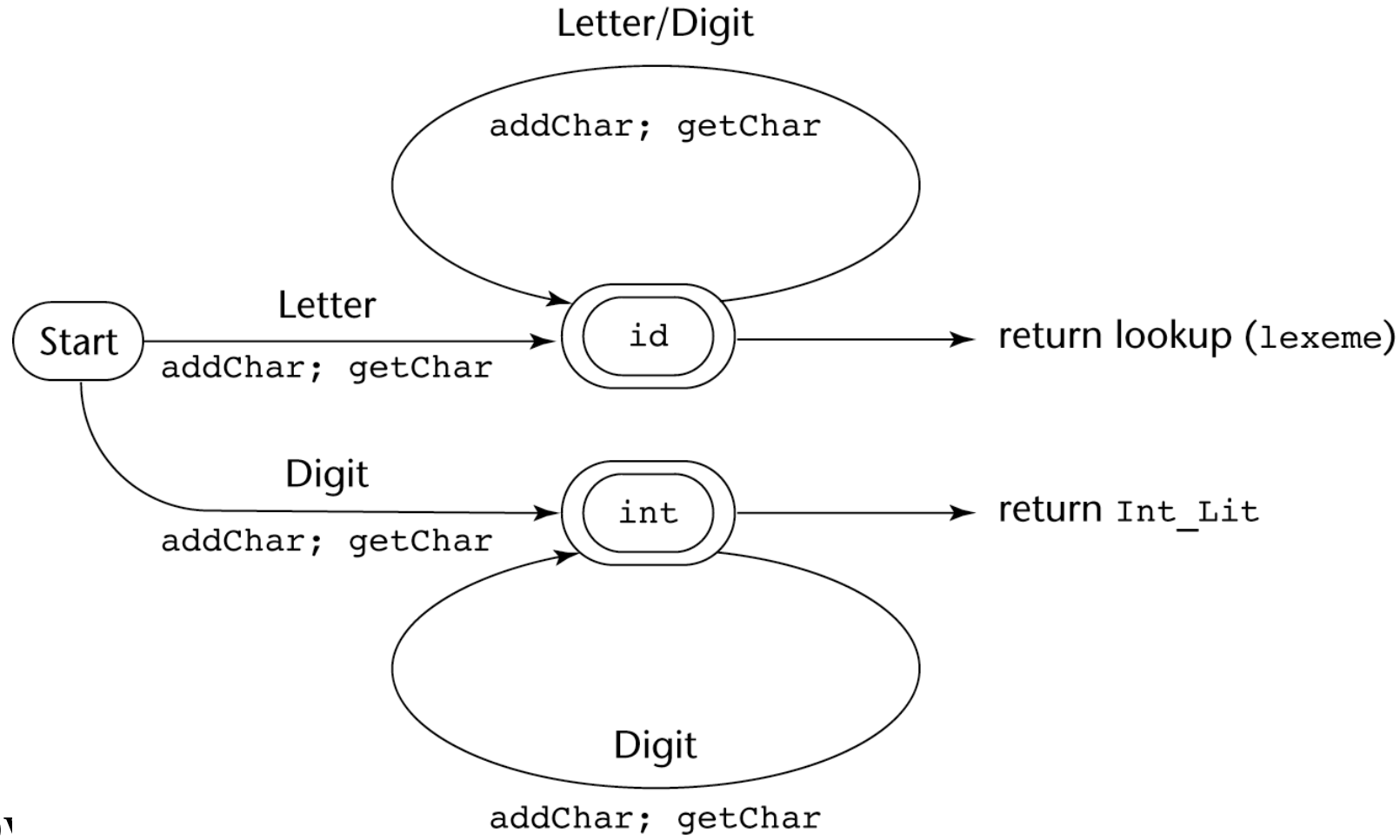
Sözcüksel Analiz

- Saklı kelimeler(Reserved words9 ve tanımlayıcılar(identifiers) birlikte tanınabilir.
  - Belirlenmiş bir tanımlayıcı(identifier) ın ayrılmış bir kelime olupo olmadığını bir tabloya baklarak karar verilebilir.

Sözcüksel Analiz

- Uygun yardımcı alt programlar:
  - **`getChar`** – girdiden bir sonraki karakteri alır, **`nextChar ın içine koyar ve sınıfını belirleyerek charClass ın içine yazar`**
  - **`addChar`** – **`nextChar`** dan karakteri lexeme içine koyar.(lexeme dizisinin sonuna ekler)
  - lookup – lexeme daki stringin bir ayrılmış kelime(reserved word) olup olmadığını belirler ve onun kodunu geriye çevirir.

Durum Diyagaramı(State Diagram)



Sözcüksel Analiz

Gerçekleştirim

```
int lex() {
  getChar();
  switch (charClass) {
    case LETTER:
      addChar();
      getChar();
      while (charClass == LETTER || charClass == DIGIT)
      {
        addChar();
        getChar();
      }
      return lookup(lexeme);
      break;
```

...

Sözcüksel Analiz

```
…
case DIGIT:
      addChar();
      getChar();
      while (charClass == DIGIT) {
        addChar();
        getChar();
      }
      return INT_LIT;
      break;
  }  /* End of switch */
}  /* End of function lex */
```

Ayrıştırma Problemi(The Parsing Problem)

- Verilen bir program için ayrıştırıcının(parser) amacı:
  - Tüm sözdizimsel hataları bulmak;her hata için iyileştirici mesajlar yayınlamak ve gerekirse düzeltmeler yapmak.
  - Bir ayrıştırma ağacı oluşturma

Ayrıştırma Problemi(The Parsing Problem)

- İki ayrıştırıcı kategorisi vardır:
  - *Yukarıdan–Aşağıya(Top down*)– ayrıştırma ağacını kökden(root) başlayarak oluşturur.
    - Sıra en sol türetme şeklindedir.
    - Ayrıştırma ağacını(parse tree) ön sıra(preorder) şeklinde oluşturur.
  - *Aşağıdan–yukarı(Bottom up)* – yapraklardan başlayarak ayrıştırma ağacını oluşturur.
    - En sağ türetme şeklinin tersi şeklinde sırası vardır.
- Ayrıştırıcı girdide her seferinde bir token a bakar.

Ayrıştırma Problemi(The Parsing Problem)

- Yukarıdan-Aşağıya Ayrıştırıcılar(Top-down Parsers)
  - Verilen bir cümlesel biçimde; xAα ,ayrıştırıcı doğru A kuralını en sol türetmedeki bir sonraki cümlesel biçimi oluşturacak şekilde seçmelidir. Bu işlemi A tarafından oluşturulan ilk token ile yapar.
- En yaygın yukarıdan aşağıya(top-down parsing) ayrıştırma algoritmaları:
  - Özyinelemeli aşağıya iniş(Recursive descent) kod temelli gerçekleştirim
  - LL ayrıştırıcılar – tablo temelli gerçekleştirim

Ayrıştırma Problemi(The Parsing Problem)

- Yukarıdan Aşağıya(Bottom-up)ayrıştırıcılar
  - Verilen bir sağ cümlesel biçimde, α, α nın hangi alt stringinin gramerdeki sağ taraf olduğunu belirler. Bu önceki sağ türetmedeki cümlesek biçimi bulmak için indirgenmelidir.
  - En yaygın yukarıdan aşağıya ayrıştırma algoritmaları (bottom-up parsing algorithms) LR family ailesindendir.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Gramerdeki her nonterminal için bir alt program vardır. Bu alt program bu nonterminal tarafından üretilen cümleleri ayrıştırır.
- EBNF bu tip ayrıştırıcılar için çok uygundur çünkü EBNF nonterminallerin sayısının mimumumlaştırır.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Bir deyim için bir gramer

```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
<factor> → id | ( <expr> )
```

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- `lex`, isminde bir sözcüksel analizcimiz olsun; bu bir sonraki tokenı(next token) `nextToken` **değişkenine koysun.**
- tek RHS olduğunda kodlama:
  - RHS deki her terminal sembolü için, bir sonraki girdi tokenı ile karşılaştır, eşleşirse devam et eşleşmezse bir hata ver
  - RHS her nonterminal sembolü için, onunla ilişkili olan alt programı çağır.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

```
/* Function expr
   Parses strings in the language
   generated by the rule:
   <expr> → <term> {(+ | -) <term>}
 */

void expr() {

/* Parse the first term */

  term();
  …
```

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

```
/* As long as the next token is + or -, call
   lex to get the next token, and parse the
   next term */

  while (nextToken == PLUS_CODE ||
         nextToken == MINUS_CODE){
    lex();
    term();
  }
}
```

- Burası hata denetimi yapmaz.
- gelenek: her ayrıştırma rutini nextToken içinde bir sonraki tokenın olmasını garanti eder.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Birden fazla RHS si olan nonterminal , hangi RHS yi ayrıştırması gerektiğine karar vermelidir.
  - Girdideki bir sonraki token a bakılarak doğru RHS e seçilir.
  - Bir sonraki token her RHS tarafından oluşturulan ilk token lar ile karşılaştırılarak doğru olan bulunur.
  - Eşleşen yoksa bir sözdizimsel hata vardır.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

```
/* Function factor
   Parses strings in the language
   generated by the rule:
   <factor> -> id  |  (<expr>)  */

 void factor() {

 /* Determine which RHS */

   if (nextToken) == ID_CODE)

 /* For the RHS id, just call lex */

     lex();
```

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

```
/* If the RHS is (<expr>) - call lex to pass
     over the left parenthesis, call expr, and
     check for the right parenthesis */

   else if (nextToken == LEFT_PAREN_CODE) {
     lex();
     expr();
     if (nextToken == RIGHT_PAREN_CODE)
       lex();
     else
       error();
   }  /* End of else if (nextToken == ...  */

   else error(); /* Neither RHS matches */
 }
```

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- LL Gramer Sınıfı(LL Grammar Class)
  - Sol özyineleme problemi(The Left Recursion Problem)
    - Eğer gramerde bir sol özyineleme varsa;ki bu direkt Ya da dolaylı özyineleme olabilir,bu yukarıdan-aşağıya ayrıştırıcı için problemdir.
      - Gramer bu sol özyinelemeyi değiştirecek şekilde değiştirilmelidir.

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Karşılıklı ayrık olmama da başka bir problemdir(pairwise disjointness)
  - Bir sonraki girdiye bakarak doğru RHS yi seçemiyorsak ayrıştırma yapmamız mümkün değildir.
  - Tanım: FIRST(α) = {a | α =>* aβ }

    (Eğer α =>* ε, ε FIRST(α) dedir)

α =>* : sıfır veya daha fazla türetme varsa

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Karşılıklı ayrık olma testi
  - Gramerdeki her birden fazla RHS si olan A nonterminali için ;her kural çifti için, $A \rightarrow \alpha_i$ veA $\rightarrow \alpha_j$, aşağıdaki sağlanmalıdır.

    $FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$
- Örnek:

  $A \rightarrow a \mid bB \mid cAb$

  $A \rightarrow a \mid aB$

Özyinelemeli Aşağıya iniş ayrıştırma(Recursive-Descent Parsing)

- Sol çarpanlarına ayırma(left factoring) bunu çözer

Yedeğiştir

<variable> → identifier | identifier [<expression>]

ile

<variable> → identifier <new>

<new> → ε | [<expression>]

yada

<variable> → identifier [[<expression>]]

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Ayrıştırmada b ir önceki sağ cümlesek biçimi elde etmek için doğru RHS yi sağ cümlesek biçimde bulmaktır.

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- handle
  - Tanım: β sağ cümlesel biçimin ***handle*** lıdır.

    $\gamma = \alpha\beta w$ sadece ve sadece $S =>^{*}rm\ \alpha Aw =>rm\ \alpha\beta w$
  - Tanım: β sağ cümlesel biçimin bir *phrase* idir.

    γ sadece ve sadece $S =>^{*} \gamma = \alpha_1 A \alpha_2 =>^{+} \alpha_1 \beta \alpha_2$
  - Tanım: β sağ cümlesel biçimin bir (*simple phrase)* idir.

    γ sadece ve sadece $S =>^{*} \gamma = \alpha_1 A \alpha_2 => \alpha_1 \beta \alpha_2$

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Handle lar hakkında:
  - Sağ cümlesek biçimin handle ı en sol simple phrase idir.
  - Verilen bir ayrıştırma ağacında bir handle bulmak oldukça basittir.
  - Ayrıştırma bir handle arama gibi algılanabilir.

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Kaydır-İndirge(Shift-Reduce) algoritmaları
  - Reduce is the action of replacing the handle on the top of the parse stack with its corresponding LHS
  - Shift is the action of moving the next token to the top of the parse stack

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- LR ayrıştırıcıların avantajları:
  - Programlama dilini tanımlayan tüm gramerler ile çalışırlar.
  - Çok geniş bir gramer sınıfı ile çalışabilir.
  - Mümkün olur olmaz sözdizimsel hataları tespit edebilir.
  - LR sınıfı gramerler LL sınıfının bir üst sınıfıdır.

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- LR ayrıştırıcılar bir araç yardımıyla oluşturulmalıdır.
- Knuth'un sezisi: Bir aşağıdan yukarıya ayrıştırıcı şu ana kadar tüm ayrıştırma geçmişini kullanarak doğru ayrıştırma kararını verebilir.
  - Göreli küçük ve belirli sayıda farklı ayrıştırma durumları olabilir. Böylece tarihçe bir ayrıştırma durumunda bir ayrıştırma yığını içinde saklanabilir.

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

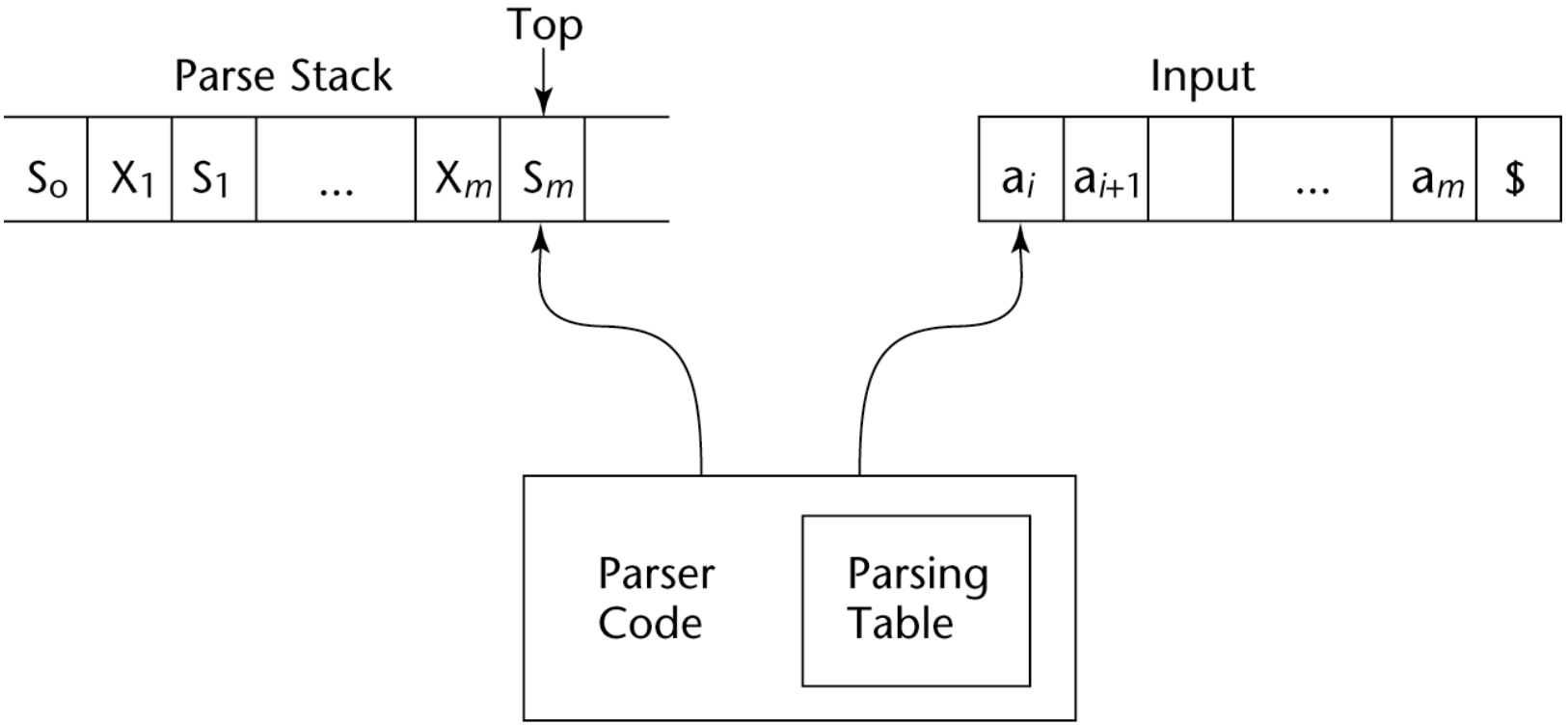
- Bir LR ayarı LR ayrıştırıcısının durumunu saklar.

$(S_0X_1S_1X_2S_2\ldots X_mS_m, a_ia_i+1\ldots a_n\$)$

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- LR ayrıştırıcılar tablo temellidir, Tablo iki bileşenlidir, bir ACTION tablosu ve bir GOTO tablosu
  - ACTION tablosu ayrıştırıcının verilen bir ayrıştırıcı durumu ve bir sonraki token a göre eylemini gösterir. Satırlar durum isimleri, sütunlar terminallerdir.
  - GOTO tablosu indirgeme işleminden sonra ayrıştırma yığınına hangi durumun koyulacağını gösterir.
    - Satırlar durum isimleri; sütunlar nonterminal lerdir.

LR ayrıştırıcısının yapısı(LR Parser structure)



Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Başlangıç ayarlar: $(S_0, a_1 \ldots a_n\$)$
- Ayrıştırıcı eylemleri:
  - Eğer ACTION$[S_m, a_i]$ = Shift S, bir sonraki ayar

    $(S_0X_1S_1X_2S_2 \ldots X_mS_ma_iS, a_{i+1} \ldots a_n\$)$
  - Eğer ACTION$[S_m, a_i]$ = Reduce $A \rightarrow \beta$ ve S = GOTO$[S_{m-r}, A]$ ise , burada r = β nin boyudur bir sonraki ayar

    $(S_0X_1S_1X_2S_2 \ldots X_{m-r}S_{m-r}AS, a_ia_{i+1} \ldots a_n\$)$

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Ayrıştırma eylemleri
  - Eğer ACTION[S_m, a_i] = Accept, ayrıştırma bitmişdir ve hata bulunmadı.
  - Eğer ACTION[S_m, a_i] = Error, ise ayrıştırıcı bir hata yönetim rutini çağırır.

LR Ayrıştırma tablosu

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Yukarıdan Aşağıya Ayrıştırma(Bottom-up Parsing)

- Bir ayrıştırma tablosu yacc benzeri programlar yardımıyla verilen gramer kullanılarak oluşturulabilir.

Özet

- Söz dizimi dil gerçekleştirimin en yaygın kısmıdır.
- Bir sözcüksel analizci(lexical analyzer) bir şablon eşleyicidir.

- Bir özyinelemeli aşağıya iniş ayrıştırıcı(recursive-descent parser) bir LL ayrıştırıcıdır(parser)
  - EBNF
- Aşağıdan yukarıya (bottom-up) ayrıştırıcıların problemi: mevcut cümlesel biçimden alt string bulmaktır.
- LR ailesi kaydır-indirge ayrıştırıcılar(shift-reduce parsers) en yaygın aşağıdan -yukarıya ayrıştırıcılardır.