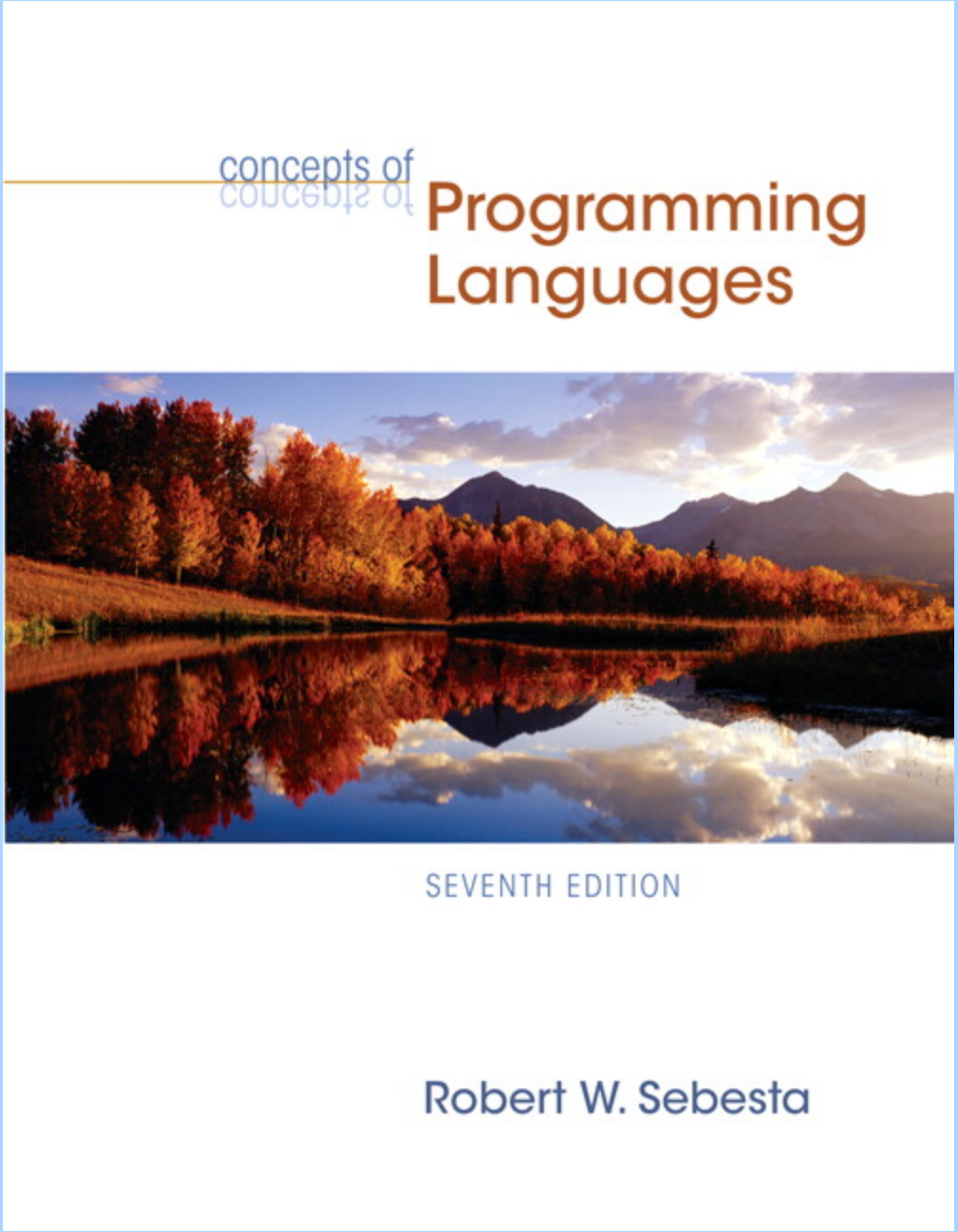
Bölüm 4

Sözcüksel(Lexical) ve Sentaks Analiz



ISBN 0-321-33025-0

Bölüm 4 Konular

1. Giriş
2. Sözcüksel Analiz(Lexical Analysis)
3. Ayrıştırma(Parsing) Problemi
4. Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)
5. Aşağıdan-yukarıya Ayrıştırma (Bottom-Up Parsing)

4.1 Giriş

- Dil(language) implementasyon sistemleri, belirli(specific) implementasyon yaklaşımına aldırmadan kaynak kodu(source code) analiz etmelidir

- Hemen hemen bütün sentaks analizi kaynak(source) dilin sentaksının biçimsel tanımlamasına(formal description) dayalıdır (BNF)

4.1 Giriş (Devamı)

- Bir dil işlemcisinin (language processor) sentaks(syntax) analizi bölümü genellikle iki kısımdan oluşur:
  - Bir düşük-düzeyli(low-level) kısım, sözcüksel analizci (lexical analyzer) (matematiksel olarak, kurallı bir gramere(regular grammar) dayalı bir sonlu otomasyon(finite automaton))
  - Bir yüksek-düzeyli(high-level) kısım, sentaks analizci(syntax analyzer), veya ayrıştırıcı(parser) (matematiksel olarak, bağlam duyarsız gramere(context-free grammar) dayalı bir aşağı-itme otomasyonu(push-down automaton), veya BNF)

4.1 Giriş (Devamı)

- Sentaksı(syntax) tanımlamak için BNF kullanmanın nedenleri :
  - Net ve özlü bir sentaks tanımı(syntax description) sağlar
  - Ayrıştırıcı(parser) doğrudan BNF ye dayalı olabilir
  - BNF ye dayalı ayrıştırıcıların(parsers) bakımı daha kolaydır

4.1 Giriş (Devamı)

- Sözcüksel(lexical) ve sentaks(syntax) analizini ayırmanın nedenleri:
  - Basitlik(Simplicity) – sözcüksel analiz (lexical analysis) için daha az karmaşık yaklaşımlar kullanılabilir; bunları ayırmak ayrıştırıcıyı(parser) basitleştirir
  - Verimlilik(Efficiency) – ayırmak sözcüksel analizcinin(lexical analyzer) optimizasyonuna imkan verir
  - Taşınabilirlik(Portability) – sözcüksel analizcinin(lexical analyzer) bölümleri taşınabilir olmayabilir, fakat ayrıştırıcı(parser) her zaman taşınabilirdir

4.2 Sözcüksel(Lexical) Analiz

- Sözcüksel analizci (lexical analyzer), karakter stringleri (character strings) için desen eşleştiricidir(pattern matcher)
- Sözcüksel analizci ayrıştırıcı(parser) için bir "ön-uç"tur ("front-end")
- Kaynak programın(source program) birbirine ait olan altstringlerini(substrings) tanımlar – lexeme'ler
  - Lexemeler, jeton(token) adı verilen sözcüksel(lexical) bir kategoriyle ilişkilendirilmiş olan bir karakter desenini eşleştirir
  - **`sum`** bir lexemedir; jetonu(token) **`IDENT`** olabilir

4.2 Sözcüksel(Lexical) Analiz (Devamı)

- Sözcüksel analizci(lexical analyzer), genellikle ayrıştırıcının sonraki jetona(token) ihtiyaç duyduğunda çağırdığı fonksiyondur. Sözcüksel analizci(lexical analyzer) oluşturmaya üç yaklaşım:
  - Jetonların biçimsel tanımı(formal description) yazılır ve bu tanıma göre tablo-sürümlü(table-driven) sözcüksel analizciyi oluşturan yazılım aracı(software tool) kullanılır
  - Jetonları(tokens) tanımlayan bir durum diyagramı(state diagram) tasarlanır ve durum diyagramını implement eden bir program yazılır
  - Jetonları(tokens) tanımlayan bir durum diyagramı(state diagram) tasarlanır ve el ile durum diyagramının(state diagram) tablo-sürümlü(table-driven) bir implementasyonu yapılır
- Sadece ikinci yaklaşımdan bahsedeceğiz

4.2 Sözcüksel(Lexical) Analiz (Devamı)

- Durum diyagramı(State diagram) tasarımı:
  - Saf(Naive) bir durum diyagramı(state diagram) kaynak(source) dildeki her karakterde her durumdan(state) bir geçişe(transition) sahip olacaktı – böyle bir diyagram çok büyük olurdu!

4.2 Sözcüksel(Lexical) Analiz (Devamı)

- Çoğu kez, durum diyagramı basitleştirmek için geçişler(transitions) birleştirilebilir
  - Bir tanıtıcıyı(identifier) tanırken, bütün büyük (uppercase) ve küçük(lowercase) harfler eşittir
    - Bütün harfleri içeren bir karakter sınıfı(character class) kullanılır
  - Bir sabit tamsayıyı (integer literal) tanırken, bütün rakamlar(digits) eşittir – bir rakam sınıfı(digit class) kullanılır

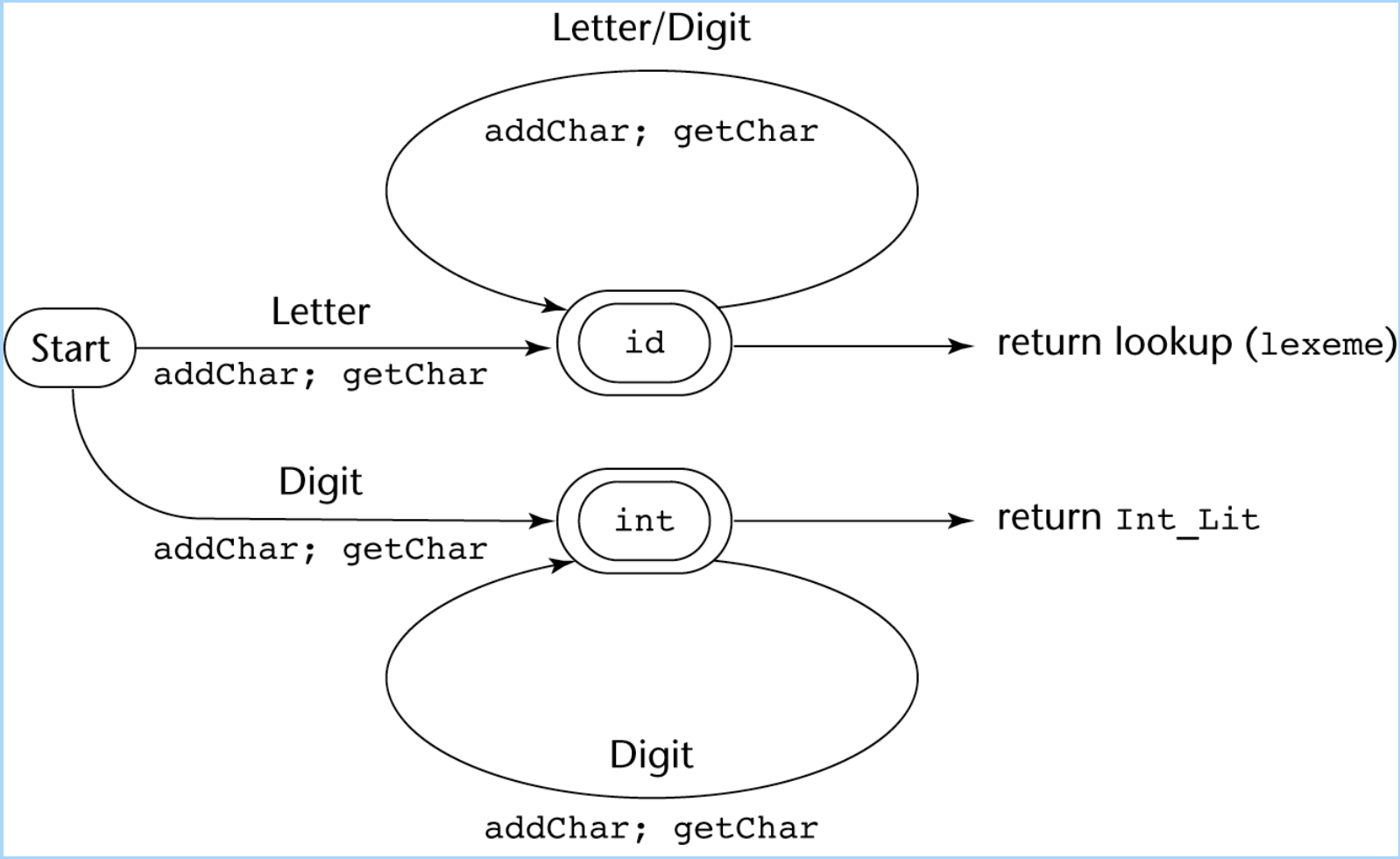
4.2 Sözcüksel(Lexical) Analiz (Devamı)

- Özgül sözcükler(reserved words) ve tanıtıcılar(identifiers) birlikte tanınabilir (her bir özgül sözcük(reserved word) için programın bir parçasını almak yerine)
  - Olası bir tanıtıcının(identifier) aslında özgül sözcük(reserved word) olup olmadığına karar vermek için tabloya başvurma(table lookup) kullanılır

4.2 Sözcüksel(Lexical) Analiz (Devamı)

- Kullanışlı yardımcı altprogramlar (utility subprograms):
  - **`getChar`** – girdinin(input) sonraki karakterini alır, bunu **`nextChar`** içine koyar, sınıfını (class) belirler ve sınıfı(class) **`charClass`** içine koyar
  - **`addChar`** – **`nextChar`** dan gelen karakteri lexemenin biriktirildiği yere koyar, **`lexeme`**
  - arama(lookup) – **`lexeme`** deki stringin özgül sözcük(reserved word) olup olmadığını belirler (bir kod döndürür)

Durum Diyagramı(State Diagram)



4.2 Sözcüksel(Lexical) Analiz (Devamı)

implementasyon (başlatma(initialization) varsayalım):

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

4.2 Sözcüksel(Lexical) Analiz(Devamı)

```
...
case DIGIT:
      addChar();
      getChar();
      while (charClass == DIGIT) {
        addChar();
        getChar();
      }
      return INT_LIT;
      break;
  }  /* switch'in sonu */
}  /* lex fonksiyonunun sonu */
```

4.3 Ayrıştırma(Parsing) Problemi

- Ayrıştırıcının amaçları, bir girdi(input) programı verildiğinde :
  - Bütün sentaks hatalarını(syntax errors) bulur; her birisi için, uygun bir tanılayıcı(diagnostic) mesaj üretir, ve hemen eski haline döndürür (recover)
  - Ayrıştırma ağacını(parse tree) üretir, veya en azından program için ayrıştırma ağacının izini(dökümünü)(trace) üretir

4.3 Ayrıştırma(Parsing) Problemi (Devamı)

- Ayrıştırıcıların(parser) iki kategorisi:
  - Aşağıdan-yukarıya(Top down) – ayrıştırma ağacını(parse tree) kökten(root) başlayarak oluşturur
    - Sıra, ensol türevindir (leftmost derivation)
    - Ayrıştırma ağacını(parse tree) preorderda izler veya oluşturur
  - Yukarıdan-aşağıya(Bottom up) – ayrıştırma ağacını(parse tree), yapraklardan(leaves) başlayarak oluşturur
    - Sıra, ensağ türevin (rightmost derivation) tersidir
- Ayrıştırıcılar(parser) girdide(input) sadece bir jeton(token) ileriye bakar

4.3 Ayrıştırma(Parsing) Problemi (Devamı)

- Yukarıdan-aşağıya ayrıştırıcılar(Top-down parsers)
  - Bir xAα sağ cümlesel formu (right sentential form) verildiğinde , ayrıştırıcı(parser), sadece A nın ürettiği ilk jetonu(token) kullanarak, ensol türevdeki(leftmost derivation) sonraki cümlesel formu(sentential form) elde etmek için doğru olan A-kuralını(A-rule) seçmelidir
- En yaygın yukarıdan-aşağıya ayrıştırma (top-down parsing) algoritmaları:
  - Özyineli azalan(recursive-descent)- kodlanmış bir implementasyon
  - LL ayrıştırıcılar(parser) – tablo sürümlü(table driven)implementasyon

4.3 Ayrıştırma(Parsing) Problemi (Devamı)

- Aşağıdan-yukarıya ayrıştırıcılar(bottom-up parsers)
  - Bir α sağ cümlesel formu(right sentential form) verildiğinde, α nın sağ türevde önceki cümlesel formu(sentential form) üretmesi için azaltılması gerekli olan, gramerde kuralın sağ tarafınd(right-hand side) olan altstringinin(substring) ne olduğuna karar verir

    (determine what substring of alfa is the right-hand side of the rule in the grammar that must be reduced to produce the previous sentential form in the right derivation)
  - En yaygın aşağıdan-yukarıya ayrıştırma(bottom-up parsing) algoritmaları LR ailesindedir

4.3 Ayrıştırma(Parsing) Problemi (Devamı)

- Ayrıştırmanın Karmaşıklığı(Complexity of Parsing)
  - Herhangi bir belirsiz-olmayan gramer(unambiguous grammar) için çalışan ayrıştırıcılar(parsers) karmaşık(complex) ve belirsizdir(inefficient) ($O(n^3)$, n girdinin(input) uzunluğu(length) olmak üzere)
  - Derleyiciler(compilers), sadece bütün belirsiz-olmayan gramerlerin(unambiguous grammars) bir altkümesi(subset) için çalışan ayrıştırıcıları(parser) kullanır, fakat bunu lineer sürede(linear time) yapar (O(n), n girdinin(input) uzunluğu(length) olmak üzere)

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)

- Özyineli-azalan işlem(Recursive-descent Process)
  - Gramerde(grammar) her bir nonterminal için o nonterminal tarafından üretilebilen cümleleri(sentences) ayrıştırabilen(parse) bir altprogram(subprogram) vardır
  - EBNF, özyineli-azalan ayrıştırıcıya (recursive-descent parser) temel oluşturmak için idealdir, çünkü EBNF nonterminal sayısını minimize eder

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing) (Devamı)

- Basit deyimler(expressions) için bir gramer(grammar) :

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

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing) (Devamı)

- **`Lex`** isimli, sonraki jeton(token) kodunu **`nextToken`** içine koyan bir anlamsal analizci(lexical analyzer) olduğunu varsayalım
- Sadece bir sağdaki kısım(RHS) olduğunda kodlama işlemi:
  - Sağdaki kısımda(RHS) olan her bir terminal sembol(symbol) için, onu bir sonraki girdi jetonuyla(token) karşılaştır; eğer eşleşiyorsa, devam et, değilse hata(error) vardır
  - Sağdaki kısımda(RHS) her bir nonterminal sembol(symbol) için, onunla ilgili ayrıştırıcı alt programını(parsing subprogram) çağırır

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

```
/* expr fonksiyonu
   Dilde kural(rule) tarafindan üretilen
  stringleri ayristirir(parses):
   <expr> → <term> {(+ | -) <term>}
 */

void expr() {

/* Ilk terimi(first term) ayristir(parse)*/

 term();
  ...
```

4.4 Özyineli-azalan Ayrıştırma(Recursive-Descent Parsing)(Devamı)

```
/* Sonraki jeton(token) + veya - oldugu sürece, sonraki
   jetonu(token) almak için lex'i çagır, ve sonraki
   terimi(next term) ayrıştır(parse)
*/

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

- Bu özel rutin hataları(errors) bulmaz
- Kural: Her ayrıştırma rutini(parsing routine) sonraki jetonu(next token) **`nextToken`** 'da bırakır

4.4 Özyineli-azalan Ayrıştırma (Recursive- Descent Parsing)(Devamı)

- Birden fazla sağdaki kısmı(RHS) olan bir nonterminal, hangi sağdaki kısmı(RHS) ayrıştıracağına(parse) karar vermek için bir başlangıç işlemine(initial process) gerek duyar
  - Doğru sağdaki kısım(RHS), girdinin(input) sonraki jetonunu(token) temel alarak seçilir (lookahead)
  - Bir eşlenik bulana kadar sonraki jeton(next token) her bir sağdaki kısım(RHS) tarafından üretilebilen ilk jetonla(first token) karşılaştırılır
  - Eğer eşlenik bulunmazsa, bu bir sentaks hatasıdır (syntax error)

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

```
/* factor fonksiyonu dilde şu kuralın(rule)
  ürettiği stringleri ayrıştırır(parse):
   <factor> -> id  |  (<expr>)  */

 void factor() {

 /* Hangi RHS oldugunu belirle*/

  if (nextToken) == ID_CODE)

 /* RHS id si için, lex 'i çagir*/

    lex();
```

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

```
/* Eğer sagdaki kısım(RHS)(<expr>) ise - sol
   parantezi ihmal ederek lex 'i çagır, expr 'yi
   çagır, ve sag parantezi kontrol et */

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

   else error(); /* Hiçbir RHS eşleşmedi*/
 }
```

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

- LL Gramer Sınıfı (LL Grammar Class)
  - Sol Özyineleme(Left Recursion) Problemi
    - Eğer bir gramerin(grammar) sol özyinelemesi(left recursion) varsa , doğrudan(direct) veya dolaylı(indirect), yukarıdan-aşağıya(Top-down) ayrıştırıcının(parser) temeli olamaz
      - Bir gramer(grammar) sol özyinelemeyi(left recursion) yoketmek için değiştirilebilir

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

- Yukarıdan-aşağıya ayrıştırmaya(top-down parsing) izin vermeyen gramerlerin diğer bir özelliği pairwise disjointness(çiftli ayrıklık) eksikliğidir
  - Doğru olan sağ kısmı(RHS) lookaheadın bir jetonuna(token) dayanarak belirleyememesi
  - Def: FIRST(α) = {a | α =>* aβ }

    (Eğer α =>* ε ise, ε FIRST(α) içindedir)

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

- Pairwise Disjointness Testi:
  - Her bir nonterminal A için, birden fazla sağ kısmı(RHS) olan gramerde(grammar), her bir kural(rule) çifti $A \rightarrow \alpha_i$ ve $A \rightarrow \alpha_j$ için, şu doğru olmalıdır:

    $$\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \phi$$

- Örnekler:

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

  $A \rightarrow a \mid aB$

4.4 Özyineli-azalan Ayrıştırma (Recursive-Descent Parsing)(Devamı)

- Sol çarpan alma(Left factoring) problemi çözebilir
  Şu ifadeyi:

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

aşağıdakilerden biriyle değiştirin:

<variable> → identifier <new>

<new> → ε | [<expression>]

veya

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

(dıştaki köşeli parantezler EBNF 'nin metasembolleridir(metasymbols))

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing)

- Ayrıştırma(parsing) problemi bir sağ-cümlesel formda(right-sentential form) türevde önceki sağ-cümlesel(right-sentential) formu elde etmek için azaltılacak doğru sağ kısmı(RHS) bulmaktır

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- İşleyiciler(tanıtıcı değer)(handles) hakkında:
  - Tanım: β , sağ cümlesel formun(right sentential form) işleyicisidir(tanıtıcı değeridir)(handle)

    $\gamma = \alpha\beta w$ ancak ve ancak (if and only if)

    $S =>^{*}rm\ \alpha Aw =>rm\ \alpha\beta w$
  - Tanım: β , sağ cümlesel formun(right sentential form) tümceciğidir(phrase)

    γ ancak ve ancak(if and only if)

    $S =>^{*} \gamma = \alpha_1 A \alpha_2 =>+ \alpha_1 \beta \alpha_2$
  - Tanım: β , sağ cümlesel form(right sentential form) γ nın basit tümceciğidir (simple phrase) ancak ve ancak(if and only if)

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

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devam)

- İşleyiciler(tanıtıcı değer)(handles) hakkında :
  - Bir sağ cümlesel formun(right sentential form) işleyicisi(handle) onun en soldaki(leftmost) basit tümceciğidir(simple phrase)
  - Verilen bir ayrıştırma ağacında(parse tree), şimdi işleyiciyi(handle) bulmak kolaydır
  - Ayrıştırma(parsing) , işleyici budama(handle pruning) olarak düşünülebilir

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing)(Devamı)

- Kaydırma-İndirgeme Algoritmaları(Shift-Reduce Algorithms)
  - İndirgeme(Reduce), ayrıştırma yığınının(parse stack) üstündeki işleyici(handle) ile ona ilişkin LHS nin yerini değiştirme işlemidir
  - Kaydırma(Shift), bir sonraki jetonu(next token) ayrıştırma yığınının(parse stack) üstüne koyma işlemidir

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing)(Devamı)

- LR ayrıştırıcıların(LR parsers) avantajları:
  - Programlama dillerini tanımlayan gramerlerin hemen hemen tümü için çalışır.
  - Diğer aşağıdan-yukarıya algoritmalardan (bottom-up algorithms) daha geniş bir gramerler sınıfı için çalışır, fakat diğer aşağıdan-yukarıya ayrıştırıcıların(bottom-up parser) herhangi biri kadar da verimlidir
  - Mümkün olan en kısa zamanda sentaks hatalarını(syntax errors) saptayabilir
  - LR gramerler sınıfı(LR class of grammars), LL ayrıştırıcıları (LL parsers) tarafından ayrıştırılabilen sınıfın(class) üstkümesidir(superset)

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- LR ayrıştırıcıları(parsers) bir araç(tool) ile oluşturulmalıdır
- Knuth'un görüşü(Knuth's insight): Bir aşağıdan-yukarıya ayrıştırıcı(bottom-up parser), ayrıştırma(parsing) kararları almak için, ayrıştırmanın o ana kadar olan bütün geçmişini(history) kullanabilirdi
  - Sonlu ve nispeten az sayıda farklı ayrıştırma durumu oluşabilirdi, bu yüzden geçmiş(history) ayrıştırma yığını(parse stack) üzerinde bir ayrıştırıcı durumunda(parser state) saklanabilirdi

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

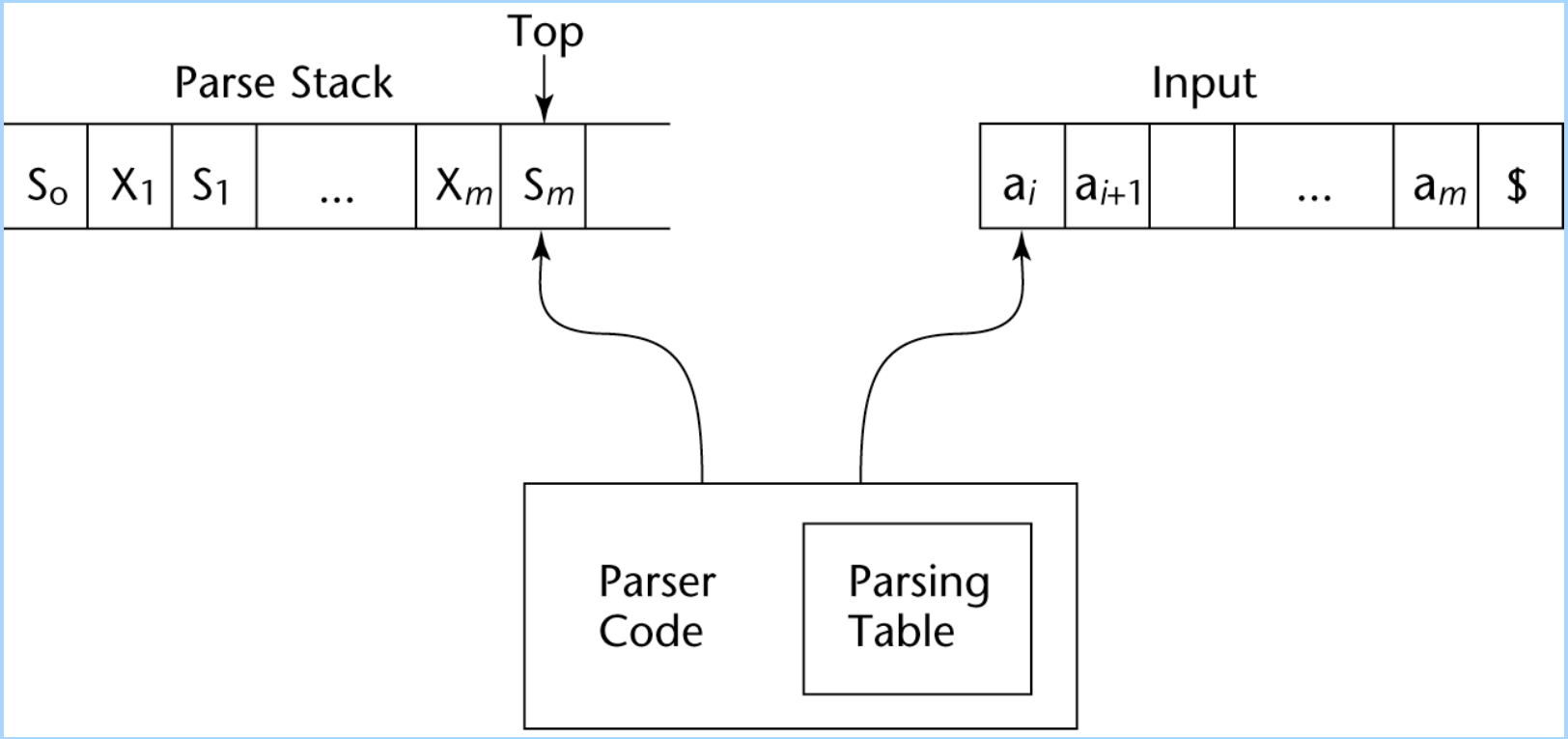
- Bir LR yapılandırması(configuration) bir LR ayrıştırıcının(LR parser) durumunu(state) saklar

- (S0X1S1X2S2…XmSm, aiai+1…an$)

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- LR ayrıştırıcılar(LR parsers) tablo sürümlüdür(table driven),bu tablonun iki bileşeni vardır, bir ACTION tablosu ve bir GOTO tablosu
  - ACTION tablosu, verilen bir ayrıştırıcı durumu(parser state) ve sonraki jeton(next token) için, ayrıştırıcının(parser) hareketini(action) belirler
    - Satırlar(rows) durum(state) adlarıdır; sütunlar(columns) terminallerdir
  - The GOTO tablosu bir indirgeme hareketi(reduction action) yapıldıktan sonra ayrıştırma yığını(parse stack) üzerine hangi durumun(state) konulacağını belirler
    - Satırlar(rows) durum(state) adlarıdır; sütunlar(columns) nonterminallerdir

Bir LR Ayrıştırıcısının(Parser) Yapısı



4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- Başlangıç yapılandırması: (S_0, $a_1 \ldots a_n\$$)
- Ayrıştırıcı hareketleri(parser actions):
  - If ACTION[S_m, a_i] = Shift S, sonraki yapılandırma:

    ($S_0X_1S_1X_2S_2 \ldots X_mS_ma_iS$, $a_{i+1} \ldots a_n\$$)
  - If ACTION[S_m, a_i] = Reduce $A \rightarrow \beta$ and S = GOTO[S_{m-r}, A], r = β nın uzunluğu olmak üzere , sonraki yapılandırma:

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

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- Ayrıştırıcı hareketleri(parser actions) (devamı):
  - If ACTION[Sm, ai] = Accept, ayrıştırma(parse) tamamlanmıştır ve hata(error) bulunmamıştır
  - If ACTION[Sm, ai] = Error, ayrıştırıcı(ayrıştırıcı) bir hata-işleme rutini(error-handling routine) çağırır

LR Ayrıştırma Tablosu(Parsing Table)

	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			

4.5 Aşağıdan-Yukarıya Ayrıştırma (Bottom-up Parsing) (Devamı)

- Verilen bir gramerden(grammar) bir araç (tool) ile bir ayrıştırıcı tablosu(parser table) üretilebilir , örn., **yacc**

Özet

- Sentaks analizi(Syntax analysis) dil implementasyonunun ortak kısmıdır
- Bir sözcüksel analizci(lexical analyzer) bir programın küçük-ölçekli parçalarını ayıran bir desen eşleştiricidir(pattern matcher)
  - Sentaks hatalarını(syntax errors) saptar
  - Bir ayrıştırma ağacı(parse tree) üretir
- Bir özyineli-iniş ayrıştırıcı(recursive-descent parser) bir LL ayrıştırıcıdır(LL parser)
  - EBNF
- Aşağıdan-yukarıya ayrıştırıcıların(bottom-up parsers) ayrıştırma problemi: o anki cümlesel formun altstringini(substring) bulma
- LR ailesi kaydırma-indirgeme ayrıştırıcıları(shift-reduce parsers) en yaygın olan aşağıdan-yukarıya ayrıştırma (bottom-up parsing) yaklaşımıdır