CENG 218 Programlama Dilleri

Bölüm 6: Sözdizimi

Öğr.Gör. Şevket Umut Çakır

Pamukkale Üniversitesi

Hafta 9

Hedefler

- Programlama dillerinin sözcüksel yapısını anlamak
- Bağlamdan bağımsız gramerleri ve BNF'leri anlamak
- Ayrıştırma ağaçları ve soyut sözdizimi ağaçlarına aşina olmak
- Belirsizliği, ilişkilendirilebilirliği ve önceliği anlamak
- EBNF'leri ve sözdizimi diyagramlarını kullanmayı öğrenmek
- Ayrıştırma tekniklerine ve araçlarına aşina olmak
- Sözcük bilgisi, sözdizimi ve semantik
- TinyAda için bir sözdizimi çözümleyicisi oluşturmak

Giriş

- Sözdizimi(Syntax), bir dilin yapısıdır
- 1950: Noam Chomsky bağlamdan bağımsız gramerler(context-free grammars) fikrini geliştirdi
- John Backus ve Peter Naur, bu gramerleri tanımlamak için şimdi Backus-Naur formları veya BNF'ler olarak adlandırılan bir notasyon sistemi geliştirdi
  - İlk olarak Algol60'ın sözdizimini tanımlamak için kullanıldı
- Her modern bilgisayar bilimcisinin, dil sözdiziminin BNF açıklamalarını nasıl okuyacağını, yorumlayacağını ve uygulayacağını bilmesi gerekir

Giriş

- Üç BNF çeşidi:
  - Orijinal BNF
  - Genişletilmiş(extended) BNF (EBNF)
  - Sözdizimi diyagramları

Programlama Dillerinin Sözcük Yapısı(Lexical Structure)

- **Sözcüksel yapı(Lexical structure)**: bir dilin sembollerinin(token) veya kelimelerinin yapısı
  - Sözdizimsel yapıyla(syntactic structure) ilgili, ancak ondan farklı
- **Tarama(Scanning)** aşaması: bir çevirmenin giriş programından karakter dizilerini topladığı ve bunları sembollere(token) dönüştürdüğü aşama
- **Ayrıştırma(Parsing)** aşaması: çevirmenin sembolleri işlediği, programın sözdizimsel yapısını belirlediği aşama

Programlama Dillerinin Sözcük Yapısı

- Semboller genellikle birkaç kategoriye ayrılır:
  - **Ayrılmış kelimeler(reserved words)** (veya **anahtar kelimeler(keywords)**)
  - **Değişmezler(Literals)** veya **sabitler(constants)**
  - ";", "<=" veya "+" gibi özel semboller
  - Tanımlayıcılar(Identifiers)
- **Önceden tanımlanmış tanımlayıcılar(Predefined identifiers)**: dildeki tüm programlar için bir başlangıç anlamı verilen ancak yeniden yönlendirme yapabilen tanımlayıcılar
- **En uzun alt dize ilkesi(Principle of longest substring)**: mümkün olan en uzun boş olmayan karakter dizisini toplama/elde etme işlemi

Programlama Dillerinin Sözcük Yapısı

- **Sembol sınırlayıcıları(Token delimiters)** (veya **beyaz boşluk(white space)**): sembollerin tanınma şeklini etkileyen biçimlendirme
- Yapıyı belirlemek için girinti(indentation) kullanılabilir
- **Serbest biçimli(Free-format)** dil: En uzun alt dize ilkesini yerine getirmek dışında formatın program yapısı üzerinde hiçbir etkisi olmayan dil
- **Sabit biçimli(Fixed-format)** dil: tüm simgelerin sayfada önceden belirlenmiş konumlarda olması gereken dil
- Semboller resmi olarak **düzenli ifadelerle(regular expressions)** tanımlanabilir

Programlama Dillerinin Sözcük Yapısı

- Düzenli ifadelerde üç temel karakter kalıbı:
  - Birleştirme(Concatenation): öğeleri sıralayarak yapılır
  - Tekrar(Repetition): tekrarlanacak öğeden sonra bir yıldız işaretiyle gösterilir
  - Seçim(Choice/Selection): seçilecek öğeler arasında dikey bir çubukla gösterilir
- [ ], kısa çizgi ile, bir dizi karakteri belirtir
- ? isteğe bağlı bir öğeyi gösterir
- Nokta herhangi bir karakteri gösterir

Programlama Dillerinin Sözcük Yapısı

- Örnekler:
  - Bir veya daha fazla basamaklı tamsayı sabitleri
    `[0-9]+`
  - İşaretsiz kayan noktalı değişmez değerler
    `[0-9]+(\.[0-9]+)?`
- Çoğu modern metin düzenleyicisi, metin aramalarında düzenli ifadeler kullanır
- **lex** gibi yardımcı programlar, bir dilin sembollerinin düzenli ifade açıklamasını otomatik olarak tarayıcıya dönüştürebilir

Programlama Dillerinin Sözcük Yapısı

- Basit tarayıcı girdisi:

  `* + ( ) 42 # 345`

- Bu çıktıyı üretir:

```
TT_TIMES
TT_PLUS
TT_LPAREN
TT_RPAREN
TT_NUMBER: 42
TT_ERROR: #
TT_NUMBER: 345
TT_EOL
```

Bağlamdan Bağımsız Gramerler ve BNF'ler

- Örnek: basit gramer

(1) *sentence* → *noun-phrase verb-phrase* .
(2) *noun-phrase* → *article noun*
(3) *article* → `a` | `the`
(4) *noun* → `girl` | `dog`
(5) *verb-phrase* → *verb noun-phrase*
(6) *verb* → `sees` | `pets`

Figure 6.2 A grammar for simple english sentences

- →: sol ve sağ tarafı ayırır
- |: bir seçimi gösterir

Bağlamdan Bağımsız Gramerler ve BNF'ler

- **Metasemboller(Metasymbols)**: gramer kurallarını açıklamak için kullanılan semboller
- Bazı gösterimlerde küçüktür ve büyüktür sembolleri ve saf metin metasembolleri kullanılır
  - Örnek: `<sentence> ::= <noun-phrase> <verb-phrase> "."`
- **Türetme(Derivation)**: bir dilde **başlangıç sembolüyle(start symbol)** başlayıp sol tarafları kurallarda sağ taraf seçimleriyle değiştirerek inşa etme süreci

Bağlamdan Bağımsız Gramerler ve BNF'ler

sentence ⇒ *noun-phrase verb-phrase* . (rule 1)
⇒ *article noun verb-phrase* . (rule 2)
⇒ `the` *noun verb-phrase* . (rule 3)
⇒ `the girl` *verb-phrase* . (rule 4)
⇒ `the girl` *verb noun-phrase* . (rule 5)
⇒ `the girl sees` *noun-phrase* . (rule 6)
⇒ `the girl sees` *article noun* . (rule 2)
⇒ `the girl sees a` *noun* . (rule 3)
⇒ `the girl sees a dog` . (rule 4)

Figure 6.3 A derivation using the grammar of Figure 6.2

Bağlamdan Bağımsız Gramerler ve BNF'ler

- Bu basit gramer ile ilgili bazı problemler:
  - Geçerli bir cümle mutlaka mantıklı olmak zorunda değildir
  - Konumsal özellikler (cümlenin başındaki büyük harf kullanımı gibi) temsil edilmez
  - Gramer, boşlukların gerekli olup olmadığını belirtmez
  - Gramer, girdi biçimini veya sonlandırma sembolünü belirtmez

Bağlamdan Bağımsız Gramerler ve BNF'ler

- Bağlamdan bağımsız gramer(Context-free grammar): bir dizi dilbilgisi kuralından oluşur
- Her kuralın solda tek bir kelime öbeği yapısı adı, ardından bir $\rightarrow$ meta simgesi, ardından sağda bir dizi sembol veya diğer kelime öbeği yapısı adları vardır
- **Terminal Olmayanlar(Nonterminals)**: İfade yapıları için isimler, çünkü bunlar başka parça yapılarına ayrılmıştır.
- **Terminaller**: Daha fazla ayrıştırılamayan kelimeler veya semboller

Bağlamdan Bağımsız Gramerler ve BNF'ler

- **Üretimler(Productions)**: gramer kuralları için başka bir isim
  - Bir bağlamdan bağımsız gramerde, terminal olmayan sayısı kadar üretim vardır
- **Backus-Naur formu(BNF)**: yalnızca "$\rightarrow$" ve "$|$" meta sembollerini kullanır
- **Başlangıç sembolü(Start symbol)**: tanımlanmakta olan üst düzey tümceciği temsil eden bir terminal olmayan
- **Gramerin dili(Language of the grammar)**: bağlamdan bağımsız bir gramer ile tanımlanır

Bağlamdan Bağımsız Gramerler ve BNF'ler

- Üretimlerin sol taraflarında tek başlarına terminal olmayan göründüğünde bir dilbilgisi bağlamdan bağımsızdır
  - Yalnızca belirli değişikliklerin yapılabileceği bir bağlam yoktur
- Bağlamdan bağımsız gramerler kullanılarak ifade edilemeyen herhangi bir şey, sözdizimsel değil anlambilimsel(semantik) bir sorundur
- BNF dil sözdizimi biçimi, çevirmen yazmayı kolaylaştırır
- Ayrıştırma aşaması otomatikleştirilebilir

Bağlamdan Bağımsız Gramerler ve BNF'ler

- Kurallar özyinelemeyi ifade edebilir

$$
\begin{aligned}
expr &\rightarrow expr + expr \mid expr * expr \mid (expr) \mid number \\
number &\rightarrow number\ digit \mid digit \\
digit &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Figure 6.4 A simple integer arithmetic expression grammar

$$
\begin{aligned}
number &\Rightarrow number\ digit \\
&\Rightarrow number\ digit\ digit \\
&\Rightarrow digit\ digit\ digit \\
&\Rightarrow 2\ \ digit\ digit \\
&\Rightarrow 23\ \ digit \\
&\Rightarrow 234
\end{aligned}
$$

Figure 6.5 A derivation for the *number* 234 using the grammar of Figure 6.4

Bağlamdan Bağımsız Gramerler ve BNF'ler

translation-unit → *external-declaration*
| *translation-unit external-declaration*

external-declaration → *function-definition* | *declaration*

function-definition → *declaration-specifiers declarator*
declaration-list compound-statement
| *declaration-specifiers declarator compound-statement*
| *declarator declaration-list compound-statement*
| *declarator compound-statement*

declaration → *declaration-specifiers* ';'
| *declaration-specifiers init-declarator-list* ';'

init-declarator-list → *init-declarator*

Figure 6.6 Partial BNFs for C (adapted from Kernighan and Ritchie [1988]) (*continues*)

Bağlamdan Bağımsız Gramerler ve BNF'ler

| *init-declarator-list* ',' *init-declarator*

init-declarator → *declarator* | *declarator* '=' *initializer*
declarator → *pointer direct-declarator* | *direct-declarator*

pointer → '*' *type-qualifier-list pointer* | '*' *type-qualifier-list*
| '*' *pointer* | '*'

direct-declarator → ID
| '(' *declarator* ')' | *direct_declarator* '[' ']'
| *direct_declarator* '[' *constant_expression* ']'
| *direct_declarator* '(' *parameter_type_list* ')'
| *direct_declarator* '(' *identifier_list* ')'
| *direct_declarator* '(' ')'
...
...

Figure 6.6 Partial BNFs for C (adapted from Kernighan and Ritchie [1988])

Ayrıştırma ve Soyut Sözdizim Ağaçları

- Sözdizimi(syntax), anlam değil yapı kurar
  - Ama anlam sözdizimiyle ilgilidir
- **Sözdizimine yönelik semantik(Syntax-directed semantics)**: bir yapının semantiğini sözdizimsel yapısıyla ilişkilendirme süreci
  - Daha sonra eklenecek semantiği yansıtacak şekilde sözdizimini oluşturmalıdır
- **Ayrıştırma ağacı(Parse tree)**: türetmede değiştirme işleminin grafiksel gösterimi

Ayrıştırma ve Soyut Sözdizim Ağaçları

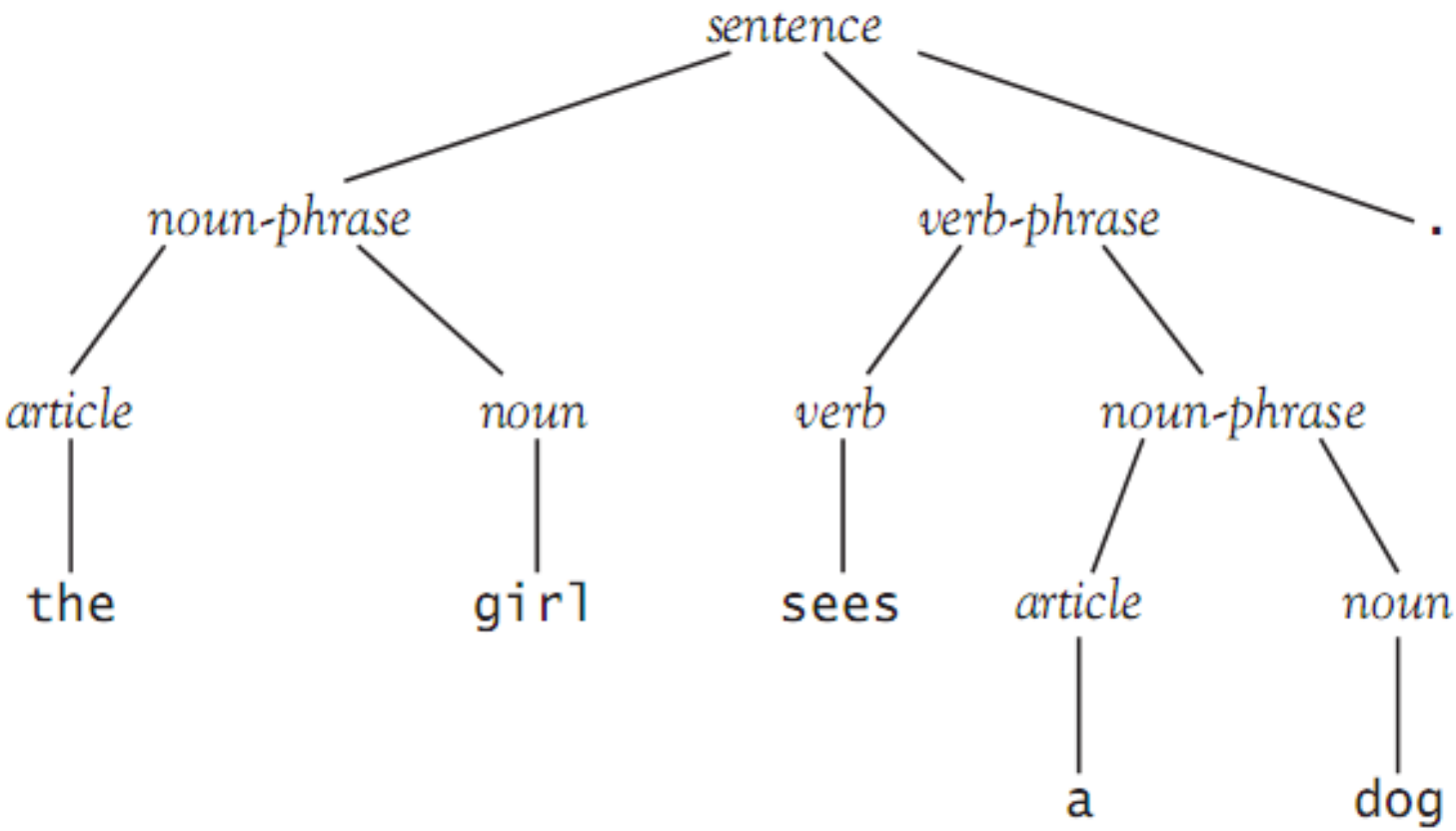


Figure 6.7: Parse tree for the sentence "`the girl sees a dog.`"

Ayrıştırma ve Soyut Sözdizim Ağaçları

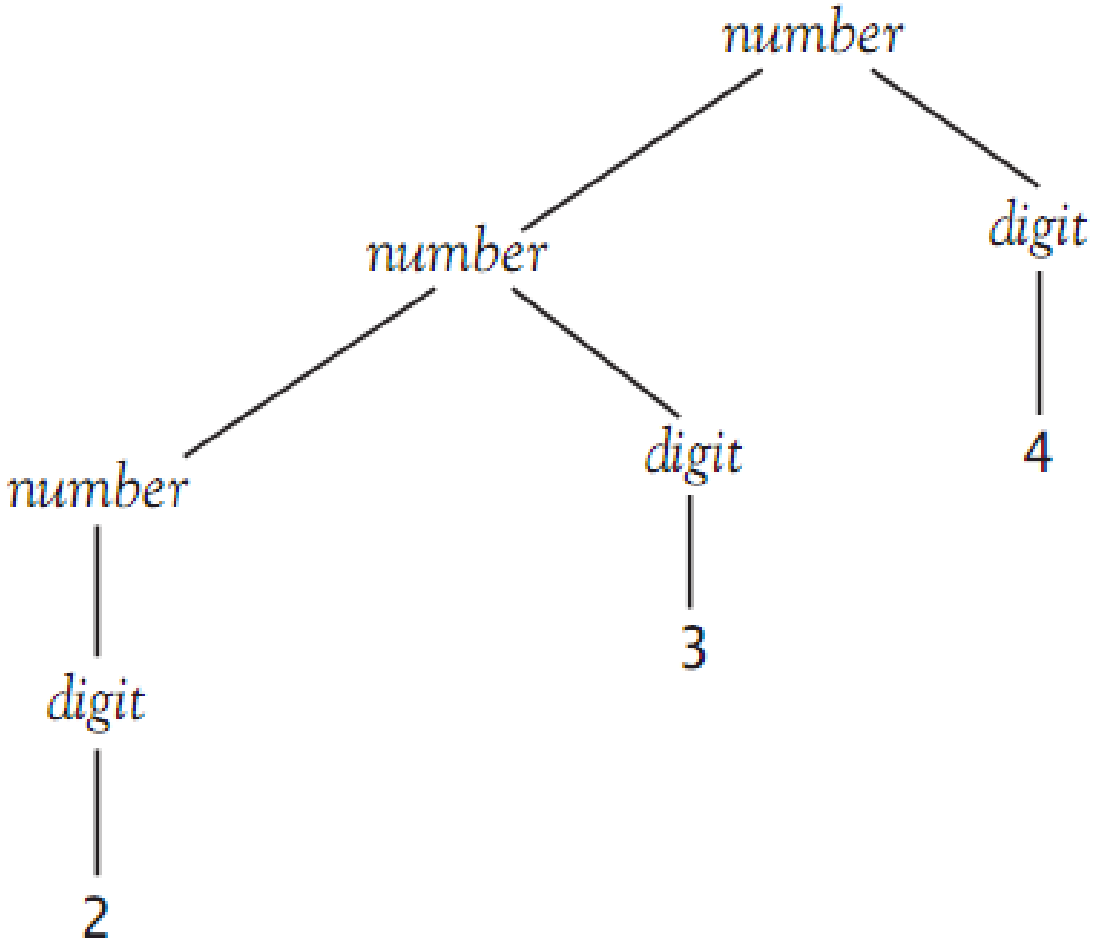


Figure 6.8: Parse tree for the number 234

Ayrıştırma ve Soyut Sözdizim Ağaçları

- En az bir çocuğu olan düğümler, terminal olmayanlarla ile etiketlenir
- Yapraklar (çocuksuz düğümler) terminallerle etiketlenir
- Bir ayrıştırma ağacının yapısı tamamen dilin gramer kuralları ve terminal dizisinin türetilmesi ile belirlenir
- Bir türetmedeki tüm terminaller ve terminal olmayanlar ayrıştırma ağacına dahil edilir

Ayrıştırma ve Soyut Sözdizim Ağaçları

- Bir ifadenin veya cümlenin sözdizimsel yapısını tam olarak belirlemek için tüm terminallere ve terminal olmayanlara gerek yoktur

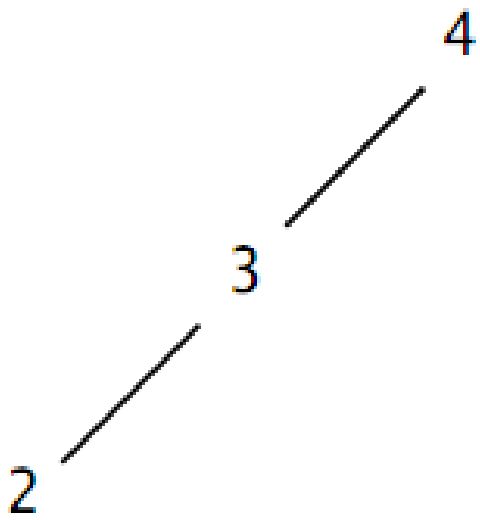


Figure 6.9: Parse tree for determining structure of the number 234

Ayrıştırma ve Soyut Sözdizim Ağaçları

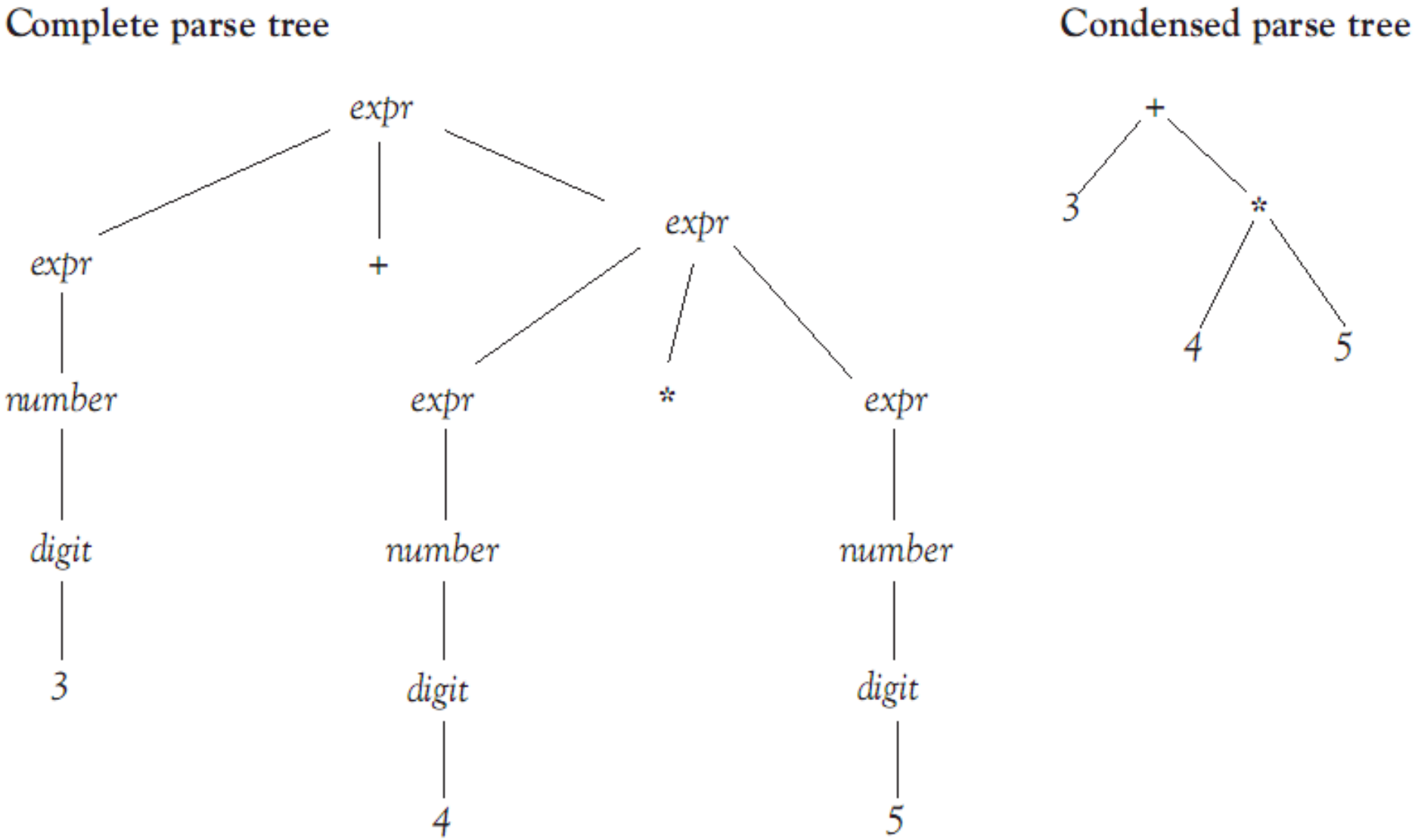


Figure 6.10: Condensing parse tree for `3 + 4 * 5`

Ayrıştırma ve Soyut Sözdizim Ağaçları

- **Soyut sözdizimi ağaçları(Abstract syntax trees)** (veya **sözdizimi ağaçları(syntax trees)**): ayrıştırma ağacının temel yapısını soyutlayan ağaçlar
  - Gereksiz terminaller ortadan kaldırılır
- Örnek:

$$\textit{if-statement} \longrightarrow \texttt{if} \ (\ \textit{expression}\ )\ \textit{statement}\ \texttt{else}\ \textit{statement}$$

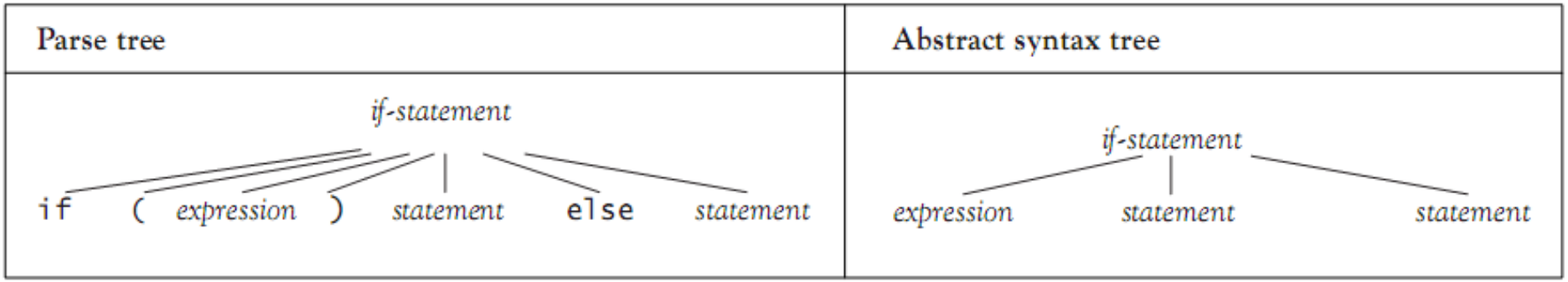


Figure 6.11: Parse tree and abstract syntax tree for grammar rule *if-statement* → `if` (*expression*) *statement* `else` *statement*

Ayrıştırma ve Soyut Sözdizim Ağaçları

- BNF kurallarına benzer soyut sözdizimi kuralları yazabilir, ancak bunlar bir programcı için daha az ilgi çekicidir
- Soyut sözdizimi, bir dil tasarımcısı ve çevirmen(derleyici/yorumlayıcı) yazanlar için önemlidir
- **Somut sözdizimi(Concrete syntax)**: sıradan sözdizimi

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

- İki farklı türetme, aynı ayrıştırma ağacına veya farklı ayrıştırma ağaçlarına yol açabilir
- **Belirsiz(Ambiguous)** gramer: iki farklı ayrıştırma veya sözdizimi ağacının mümkün olduğu gramer
- Örnek: daha önce verilen 234 için türetme

$$
\begin{aligned}
number &\Rightarrow number\ digit \\
&\Rightarrow number\ 4 \\
&\Rightarrow number\ digit\ 4 \\
&\Rightarrow number\ 34 \\
&\ldots
\end{aligned}
$$

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

$$
\begin{aligned}
expr &\rightarrow expr\ +\ expr \mid expr\ *\ expr \mid (\ expr\ ) \mid number \\
number &\rightarrow number\ digit \mid digit \\
digit &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Figure 6.4 A simple integer arithmetic expression grammar

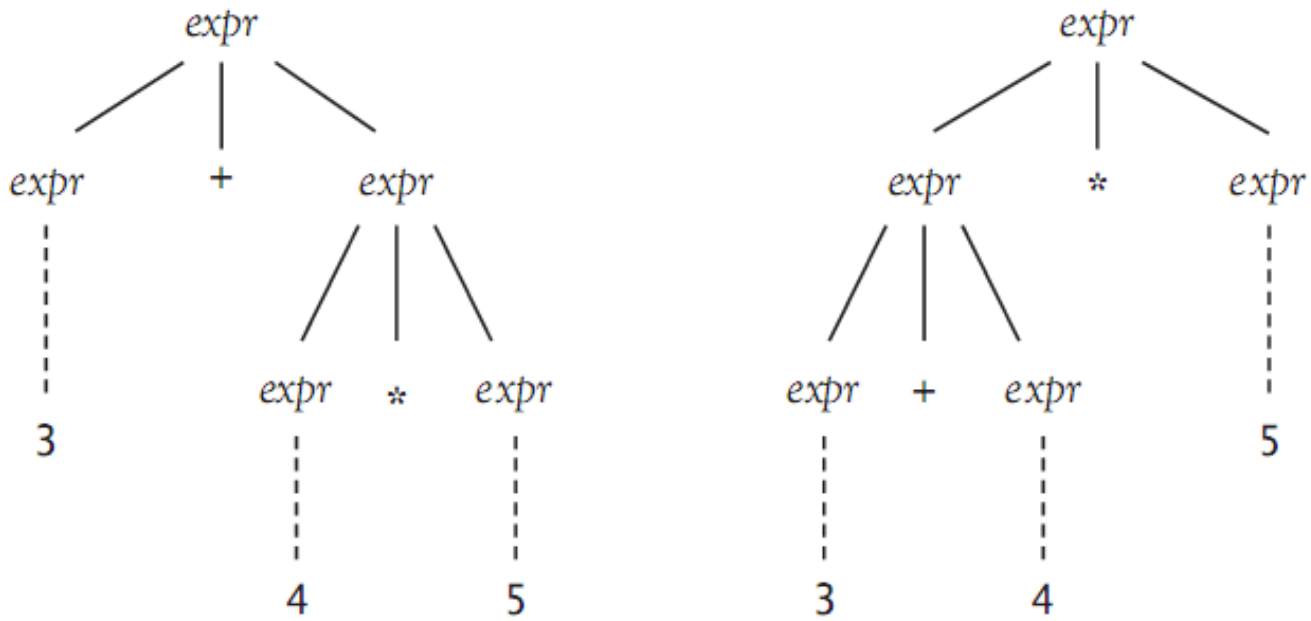


Figure 6.12: Two parse trees for `3 + 4 * 5`

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

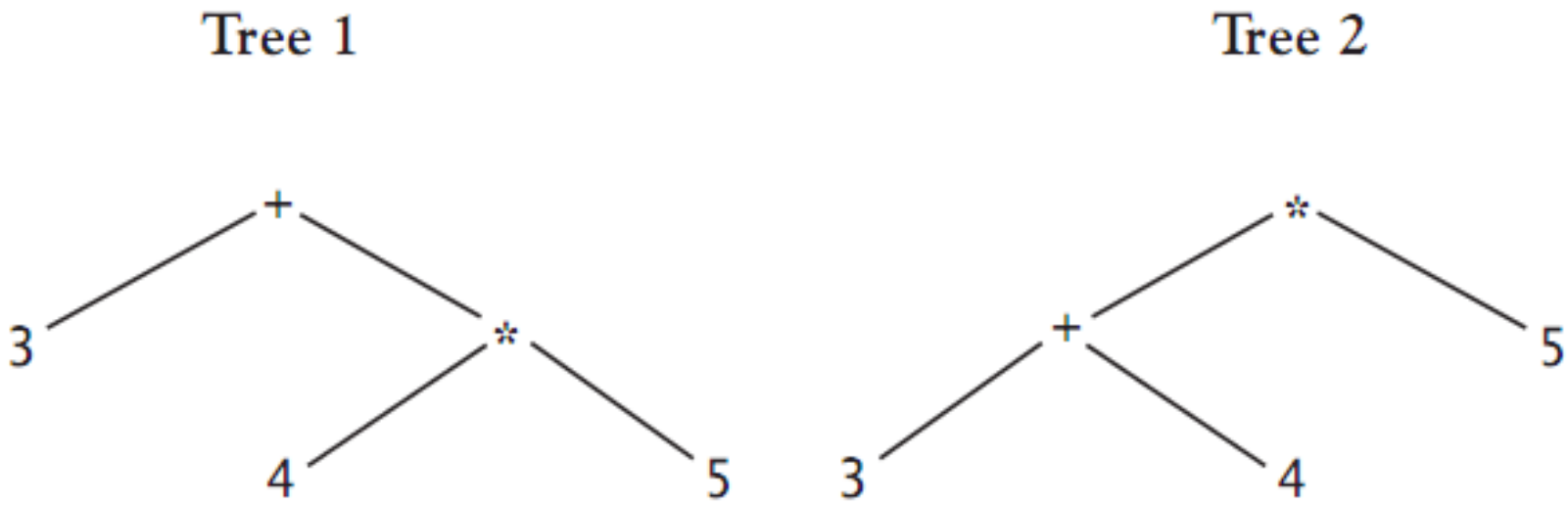


Figure 6.13 Two abstract syntax trees for `3 + 4 * 5`, indicating the ambiguity of the grammar of Figure 6.4

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

- Özel bir sırayla inşa edilen belirli özel türetmeler yalnızca benzersiz ayrıştırma ağaçlarına karşılık gelebilir
- **En soldaki türetme(Leftmost derivation)**: en solda kalan terminal olmayan(nonterminal) her adımda değiştirilmek üzere ayrılır
  - Her ayrıştırma ağacının benzersiz bir en soldaki türetmesi vardır
- Bir gramerin belirsizliği(ambiguity), en soldaki iki farklı türetme aranarak test edilebilir

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

Leftmost Derivation 1 (Corresponding to Tree 1 of Figure 6.13)	**Leftmost Derivation 2 (Corresponding to Tree 2 of Figure 6.13)**
$expr \Rightarrow expr + expr$	expr $\Rightarrow$ expr * $expr$
$\Rightarrow number + expr$	$\Rightarrow expr + expr * expr$
$\Rightarrow digit + expr$	$\Rightarrow number + expr * expr$
$\Rightarrow 3 + expr$	$\Rightarrow$ . . . (etc.)
$\Rightarrow 3 + expr * expr$	
$\Rightarrow 3 + number * expr$	
$\Rightarrow$ . . . (etc.)	

Figure 6.14 Two leftmost derivations for `3 + 4 * 5`, indicating the ambiguity of the grammar of Figure 6.4

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

- Belirsiz gramerler zorluklar yaratır
  - Belirsizliği gidermek için gramer revize edilmeli veya **belirsizliği ortadan kaldıran bir kural(disambiguating rule)** belirtmelidir
- Grameri revize etmenin genel yolu, bir öncelik sırasını belirleyen bir terim adı verilen yeni bir dilbilgisi kuralı yazmaktır
- $expr \rightarrow expr + expr$ ifadesi
  - $expr \rightarrow expr + term$ ile veya $expr \rightarrow term + expr$ ile yer değiştirebilir
- İlk kural sol özyinelemelidir; ikinci kural sağ özyinelemelidir

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

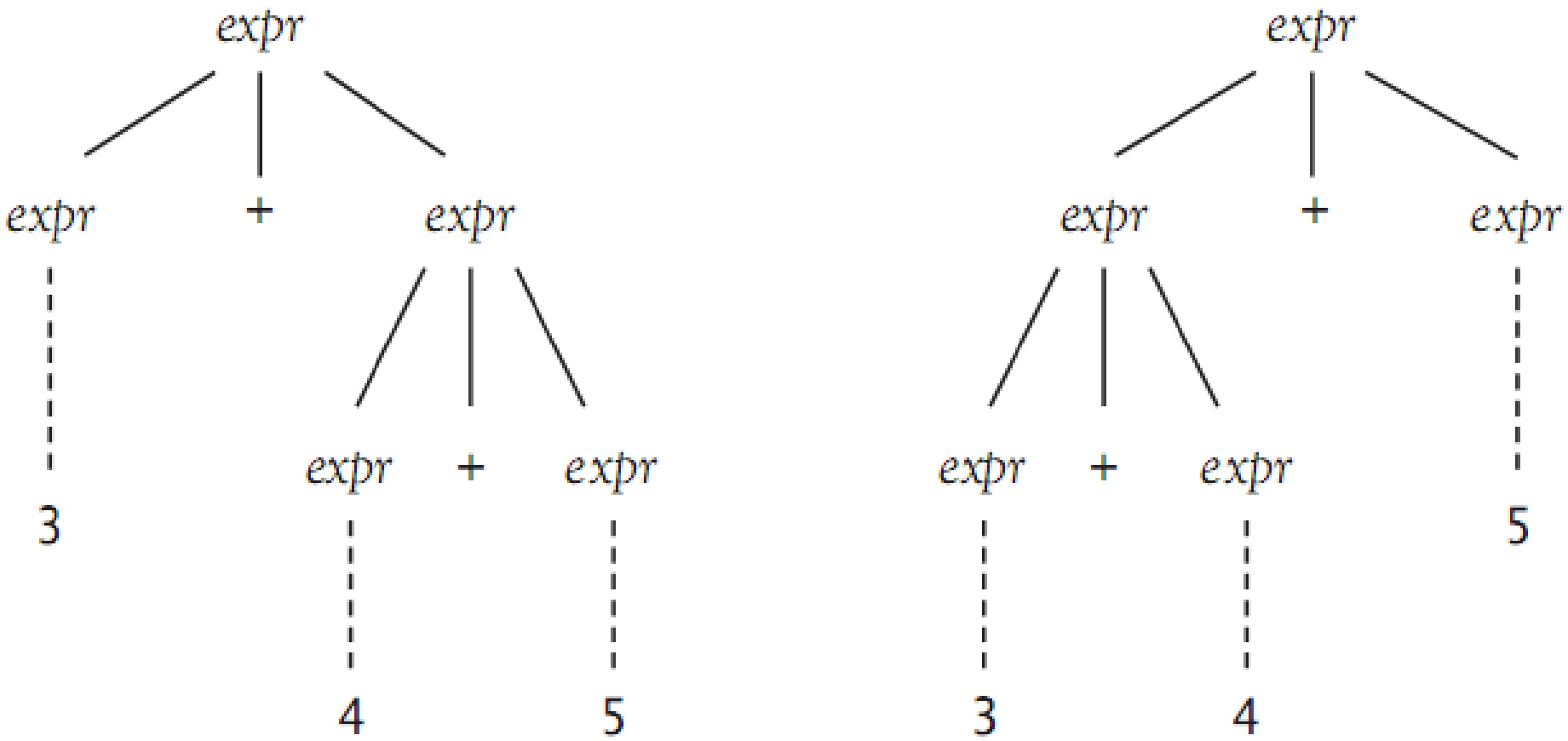


Figure 6.15: Addition as either right- or left-associative

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

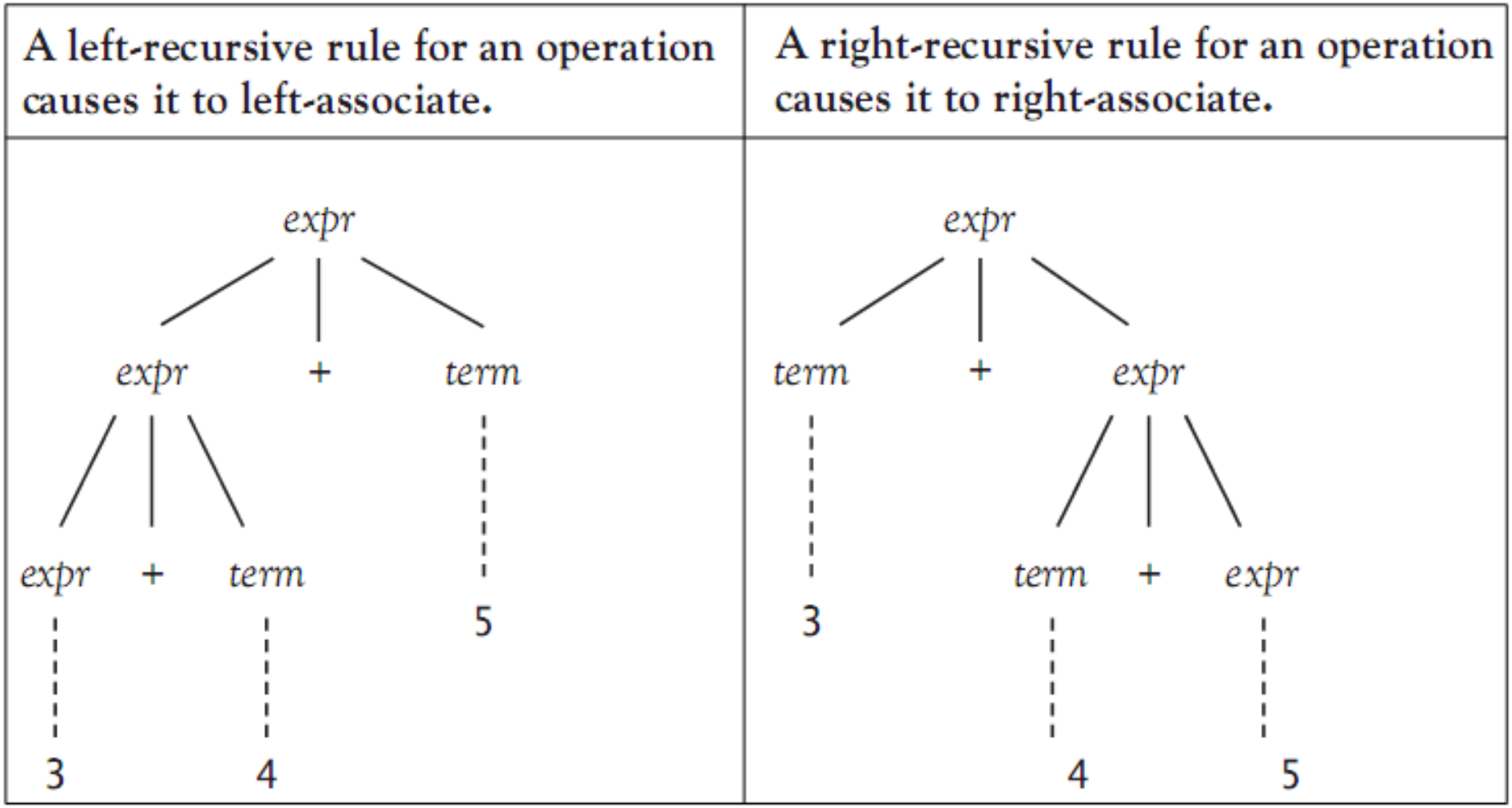


Figure 6.16: Parse trees showing results of left- and right-recursive rules

Belirsizlik, İlişkilendirilebilirlik ve Öncelik

$$
\begin{aligned}
expr &\longrightarrow expr \ + \ term \mid term \\
term &\longrightarrow term \ * \ factor \mid factor \\
factor &\longrightarrow (\ expr \ ) \mid number \\
number &\longrightarrow number \ digit \mid digit \\
digit &\longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Figure 6.17 Revised grammar for simple integer arithmetic expressions

EBNF'ler ve Sözdizimi Diyagramları

- **Genişletilmiş(Extended) Backus-Naur formu** (veya EBNF): yaygın sorunları ele almak için yeni gösterim sunar
- 0 veya daha fazla tekrarı belirtmek için süslü parantez kullanılır
  - Küme parantez tekrarına dahil olan herhangi bir operatörün sola ilişkilendirilmiş olduğunu varsayar
  - Örnek: $number \rightarrow digit\{digit\}$
- İsteğe bağlı parçaları belirtmek için köşeli parantez kullanılır
  - Örnek: $if\text{-}statement \rightarrow$ if $(expression)\ statement[$else $statement]$

EBNF'ler ve Sözdizimi Diyagramları

$$
\begin{aligned}
&\textit{expr} \longrightarrow \textit{term} \ \{ \ + \ \textit{term} \ \} \\
&\textit{term} \longrightarrow \textit{factor} \ \{ \ * \ \textit{factor} \ \} \\
&\textit{factor} \longrightarrow \ (\ \textit{expr} \ ) \ \mid \ \textit{number} \\
&\textit{number} \longrightarrow \textit{digit} \ \{ \ \textit{digit} \ \} \\
&\textit{digit} \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Figure 6.18 EBNF rules for simple integer arithmetic expressions

EBNF'ler ve Sözdizimi Diyagramları

- **Sözdizimi diyagramı(Syntax diagram)**: kuralın sağ tarafında karşılaşılan terminallerin ve terminal olmayanların sırasını gösterir

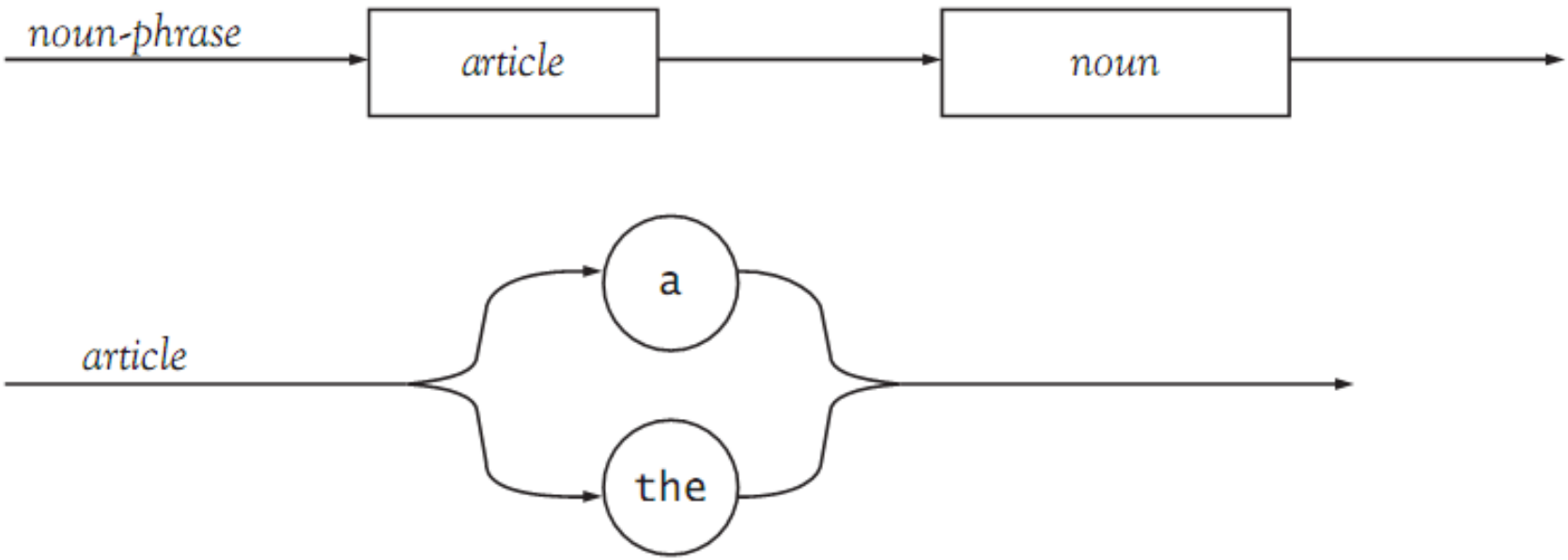


Figure 6.19: Syntax diagrams for *noun-phrase* and *article* of the simple English grammar presented in Section 6.2

EBNF'ler ve Sözdizimi Diyagramları

- Terminaller için daire veya oval, terminal olmayanlar için kare veya dikdörtgen kullanılır
  - Bunlar uygun sıralamayı gösteren çizgiler ve oklarla bağlanır
- Birkaç kuralı tek bir diyagramda toplayabilir
- Tekrarı belirtmek için döngüler kullanılır

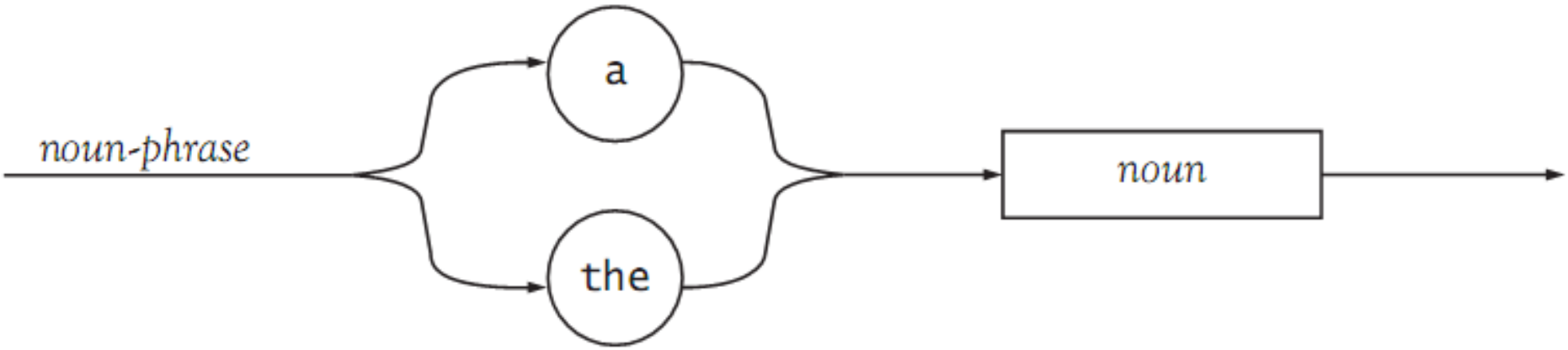
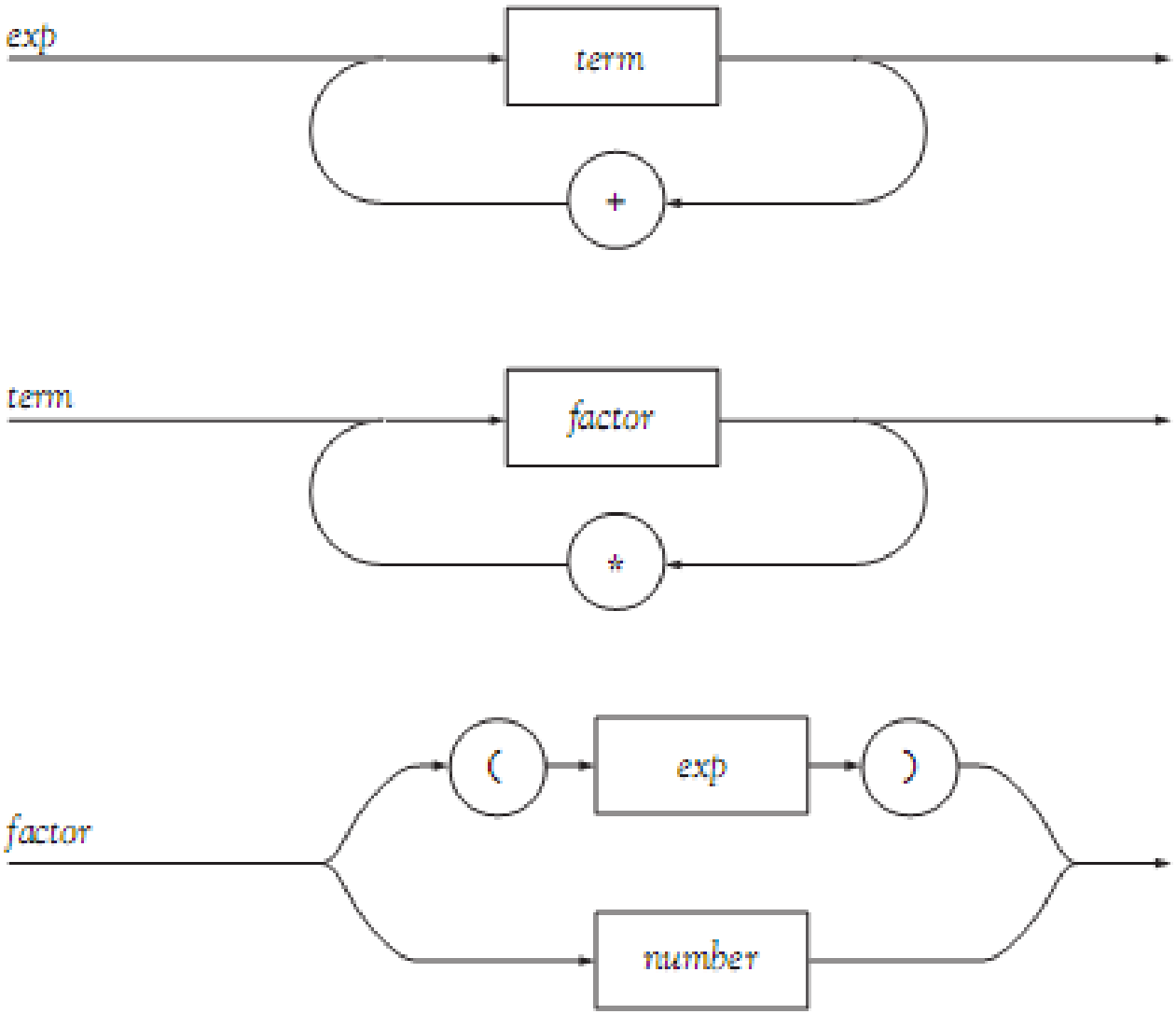


Figure 6.20: Condensed version of diagrams shown in Figure 6.19

EBNF'ler ve Sözdizimi Diyagramları



EBNF'ler ve Sözdizimi Diyagramları

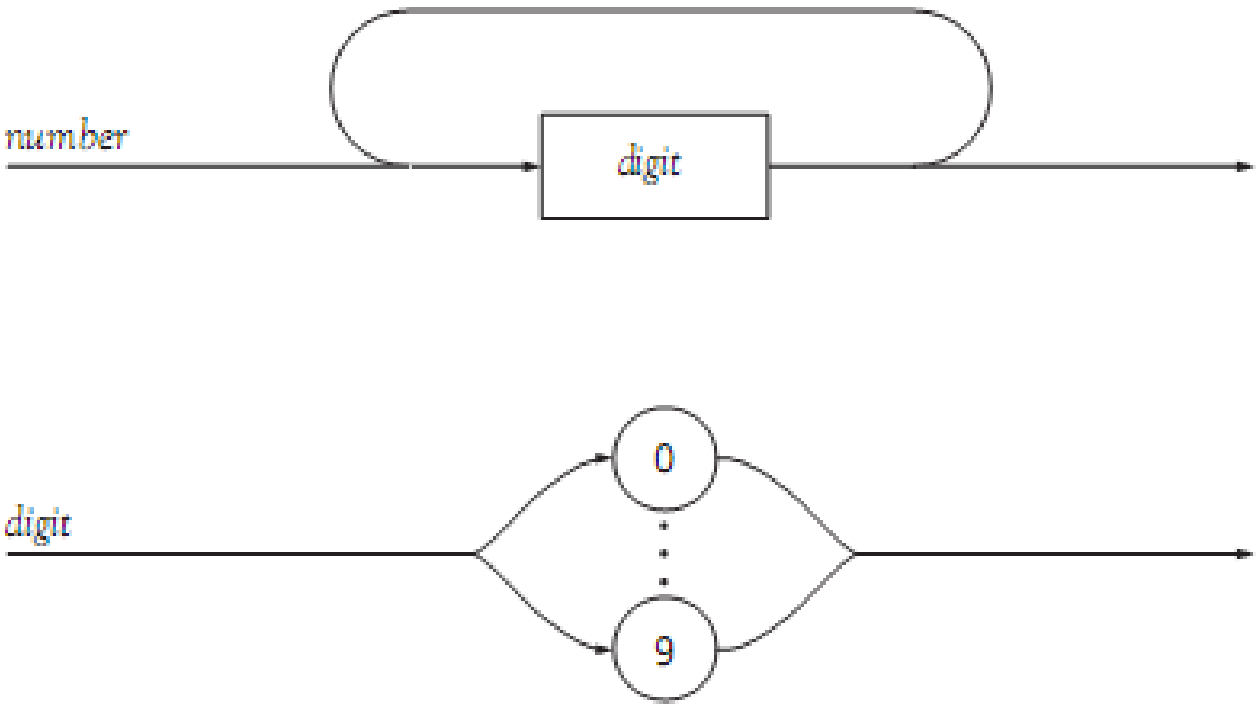


Figure 6.21: Syntax diagrams for a simple integer expression grammar

EBNF'ler ve Sözdizimi Diyagramları

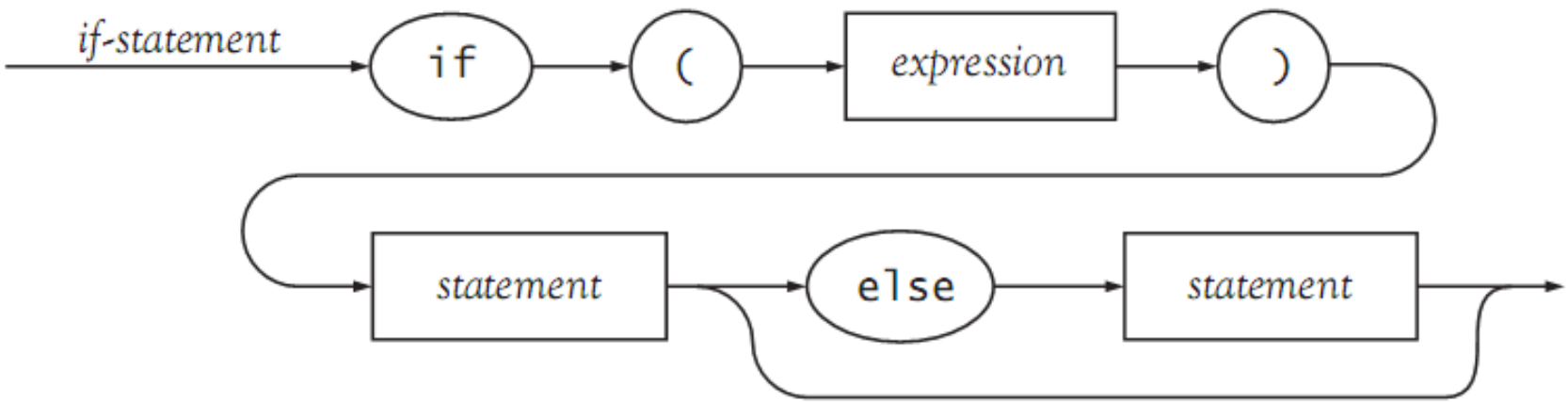


Figure 6.22 Syntax diagram for the *if-statement* in C

Ayrıştırma Teknikleri ve Araçları

- BNF, EBNF veya sözdizimi diyagramlarında yazılmış bir gramer, sözdizimsel olarak geçerli olan simge dizilerini açıklar
  - Ayrıca, bir ayrıştırıcının doğru şekilde ayrıştırmak için nasıl davranması gerektiğini de açıklar
- **Tanıyıcı(Recognizer)**: dilde geçerli dizeler(legal strings) olup olmadıklarına bağlı olarak dizeleri kabul eder veya reddeder
- **Aşağıdan yukarıya ayrıştırıcı(Bottom-up parser)**: türetmeler oluşturur ve ağaçları yapraklardan köklere kadar ayrıştırır
  - Girdiyi bir kuralın sağ tarafıyla eşleştirir ve sol taraftaki terminale **indirger(reduces)**

Ayrıştırma Teknikleri ve Araçları

- Aşağıdan yukarı ayrıştırıcılara ayrıca kaydırma azaltma(shift-reduce) ayrıştırıcıları da denir
  - Dizeleri terminal olmayanlara indirgemeden önce sembolleri bir yığına kaydırırlar
- **Yukarıdan aşağıya ayrıştırıcı(Top-down parser)**: Terminal olmayanları gelen sembollerle eşleşecek şekilde genişletir ve doğrudan bir türetme oluşturur
- **Ayrıştırıcı oluşturucu(Parser generator)**: yukarıdan aşağıya veya aşağıdan yukarıya ayrıştırmayı otomatikleştiren bir program
- Aşağıdan yukarıya ayrıştırma, ayrıştırıcı oluşturucular için tercih edilen yöntemdir (**derleyici derleyicileri(compiler compiler)** olarak da adlandırılır)

Ayrıştırma Teknikleri ve Araçları

- **Özyinelemeli iniş ayrıştırma**: terminal olmayanları, BNF'lerin sağ taraflarına dayalı olarak karşılıklı olarak yinelemeli yordamlar grubuna dönüştürür
  - Semboller, bir tarayıcı tarafından oluşturulan giriş sembolleriyle doğrudan eşleştirilir
  - Terminal olmayanlar, terminal olmayanlara karşılık gelen prosedürlere yapılan çağrılar olarak yorumlanır

Ayrıştırma Teknikleri ve Araçları

```
void sentence(){
        nounPhrase();
        verbPhrase();
}
void nounPhrase(){
        article();
        noun();
}
void article(){
        if (token == "a") match("a", "a expected");
        else if (token == "the") match("the", "the expected");
        else error("article expected");
}
void match(TokenType expected, char * message){
        if (token == expected) getToken();
        else error(message);
}
```

Ayrıştırma Teknikleri ve Araçları

- Sol özyinelemeli kurallar sorun yaratabilir
  - Örnek: $expr \rightarrow expr + term \mid term$
  - Sonsuz bir özyinelemeli döngüye neden olabilir
  - Bir $+$ görünene kadar iki seçenekten hangisinin alınacağına karar vermenin yolu yok
- EBNF tanımı özyinelemeyi bir tekrarlama olarak ifade eder: $expr \rightarrow term\{+term\}$
- Bu nedenle, EBNF'deki süslü parantezler, bir döngü kullanılarak sol özyinelemeli kaldırmayı temsil eder.

Ayrıştırma Teknikleri ve Araçları

- Aşağıdakiler gibi, doğru özyinelemeli bir kural için kod:
  $expr \rightarrow term@expr \mid term$
- Bu, EBNF'de köşeli parantez kullanımına karşılık gelir:
  $expr \rightarrow term[@expr]$
  - Bu işleme **sol faktörleme(left factoring)** denir
- Hem sol özyinelemeli hem de sol faktörleme durumlarında, EBNF kuralları veya sözdizimi diyagramları doğal olarak özyinelemeli bir ayrıştırıcının koduna karşılık gelir

Ayrıştırma Teknikleri ve Araçları

- **Tek sembollü önden okuma(Single-symbol lookahead)**: bir ayrıştırmayı yönlendirmek için tek bir sembol kullanma **Tahmine dayalı ayrıştırıcı(Predictive parser)**: kendisini yalnızca önden okumaya dayalı olarak belirli bir eyleme adayan bir ayrıştırıcı
- Bu karar verme sürecinin işlemesi için gramer belirli koşulları sağlamalıdır
  - Ayrıştırıcı, bir kuraldaki seçimler arasında ayrım yapabilmelidir
  - İsteğe bağlı bir parça için, isteğe bağlı parçadan sonra hiçbir sembol gelemez

Ayrıştırma Teknikleri ve Araçları
YACC

- YACC: yaygın olarak kullanılan bir ayrıştırıcı oluşturucu
  - Ücretsiz sürüm Bison olarak adlandırılır
  - Grameri ayrıştırmak için aşağıdan yukarıya bir algoritma kullanan bir C programı oluşturur
- YACC, gramerden, bir `main` prosedüründen çağrılması gereken, `yyparse` adında bir prosedür oluşturur.
- YACC, sembollerin `yylex` adındaki tarayıcı prosedürü tarafından tanındığını varsayar

Ayrıştırma Teknikleri ve Araçları

ANTLR

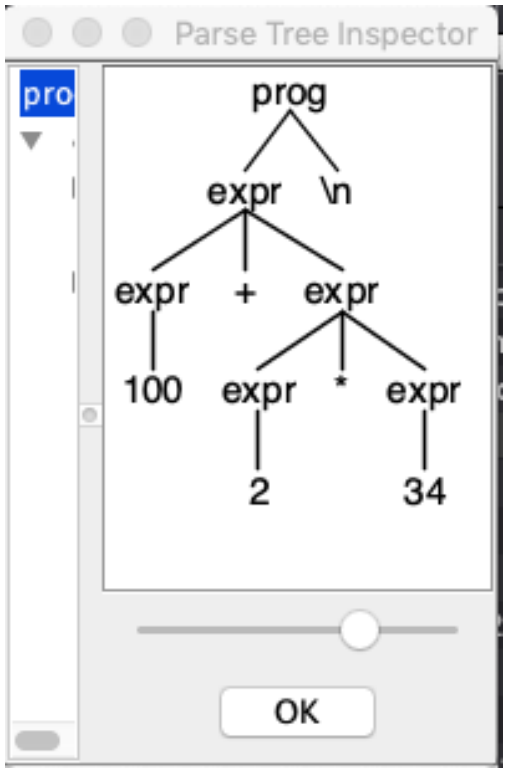
- ANTLR(ANother Tool for Language Recognition): Yapısal metin veya ikili dosyaları okumak, işlemek, çalıştırmak veya çevirmek için kullanılan bir ayrıştırıcı oluşturucu
- Bir çok hedef dile çıktı verebilir: C++, Java, C#, Pyhton, Javascript, Swift, Go, PHP, Dart
- `https://github.com/sevketcakir/algoritmapy` adresinde ANTLR ve Python ile yazılmış bir algoritma yorumlayıcı bulunmaktadır

Ayrıştırma Teknikleri ve Araçları

ANTLR

```
grammar Expr;
prog: (expr NEWLINE)* ;
expr: expr ('*'|'/') expr
    | expr ('+'|'-') expr
    | INT
    | '(' expr ')'
    ;
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

```
$ antlr4 Expr.g4
$ javac Expr*.java
$ grun Expr prog -gui
100+2*34
^D
```



Sözcük bilgisi, Sözdizimi ve Semantik

- Beyaz boşluk kuralları gibi belirli biçimlendirme ayrıntıları tarayıcıya bırakılmıştır
  - Dilbilgisinden ayrı sözcük kuralları olarak belirtilmesi gerekir
- Bir tarayıcının değişmez değerler, sabitler ve tanımlayıcılar gibi yapıları tanımasına izin verilmesi de istenir
  - Daha hızlı, daha basit ve ayrıştırıcının boyutunu küçültür
- Terminal olmayan bir temsilden ziyade bir sembol kullanımını ifade etmek için gramer yeniden yazılmalıdır

Sözcük bilgisi, Sözdizimi ve Semantik

- Örnek: bir sayı bir sembol olmalıdır

$$
\begin{aligned}
expr &\rightarrow term \ \{ \ + \ term \} \\
term &\rightarrow factor \ \{ \ * \ factor \ \} \\
factor &\rightarrow \ (\ expr \ ) \ \mid \ \texttt{NUMBER}
\end{aligned}
$$

Figure 6.26 Numbers as tokens in simple integer arithmetic

  - Büyük harf, bunun, yapısı tarayıcı tarafından belirlenen bir sembol olduğunu belirtir

- **Sözlükler(Lexics)**: bir programlama dilinin sözcüksel yapısı

Sözcük bilgisi, Sözdizimi ve Semantik

- Bazı kurallar bağlama duyarlıdır ve bağlamdan bağımsız kurallar olarak yazılamaz
- Örnekler:
  - Değişkenler için kullanımdan önce beyan(declaration before use)
  - Bir prosedür içinde tanımlayıcıların yeniden beyan edilmemesi
- Bunlar bir dilin anlamsal(semantik) özellikleridir
- Önceden tanımlanmış tanımlayıcılar ve ayrılmış sözcükler arasında başka bir çakışma meydana gelir
  - Ayrılmış kelimeler tanımlayıcı olarak kullanılamaz
  - Önceden tanımlanmış tanımlayıcılar bir programda yeniden tanımlanabilir
- Sözdizimi ve semantik, belirsiz ayrıştırma durumlarını ayırt etmek için anlamsal bilgi gerektiğinde birbirine bağımlı hale gelebilir

Örnek Olay: TinyAda için Sözdizimi Çözümleyicisi Oluşturma

- **TinyAda**: birçok üst düzey dilin sözdizimsel özelliklerini gösteren küçük bir dil
- TinyAda çeşitli türden beyanlar, ifadeler ve ifadeler içerir
- Bildirimler, ifadeler ve komutlar için kurallar dolaylı olarak özyinelemelidir, iç içe bildirimler, ifadeler ve komutlar için izin verilir
- **Ayrıştırma kabuğu(Parsing shell)**: simgelerin doğru türde olup olmadığını kontrol etmek için dilbilgisi kurallarını uygular
  - Daha sonra anlamsal analiz için mekanizmalar ekleyeceğiz

Örnek Olay: TinyAda için Sözdizimi Çözümleyicisi Oluşturma

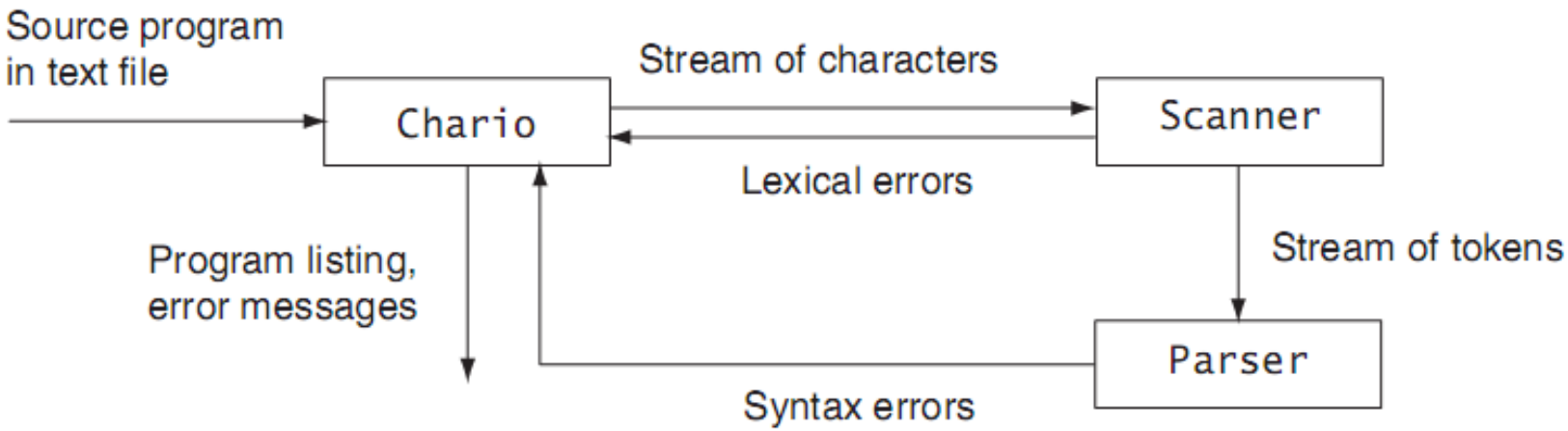


Figure 6.29 Data flow in the TinyAda syntax analyzer