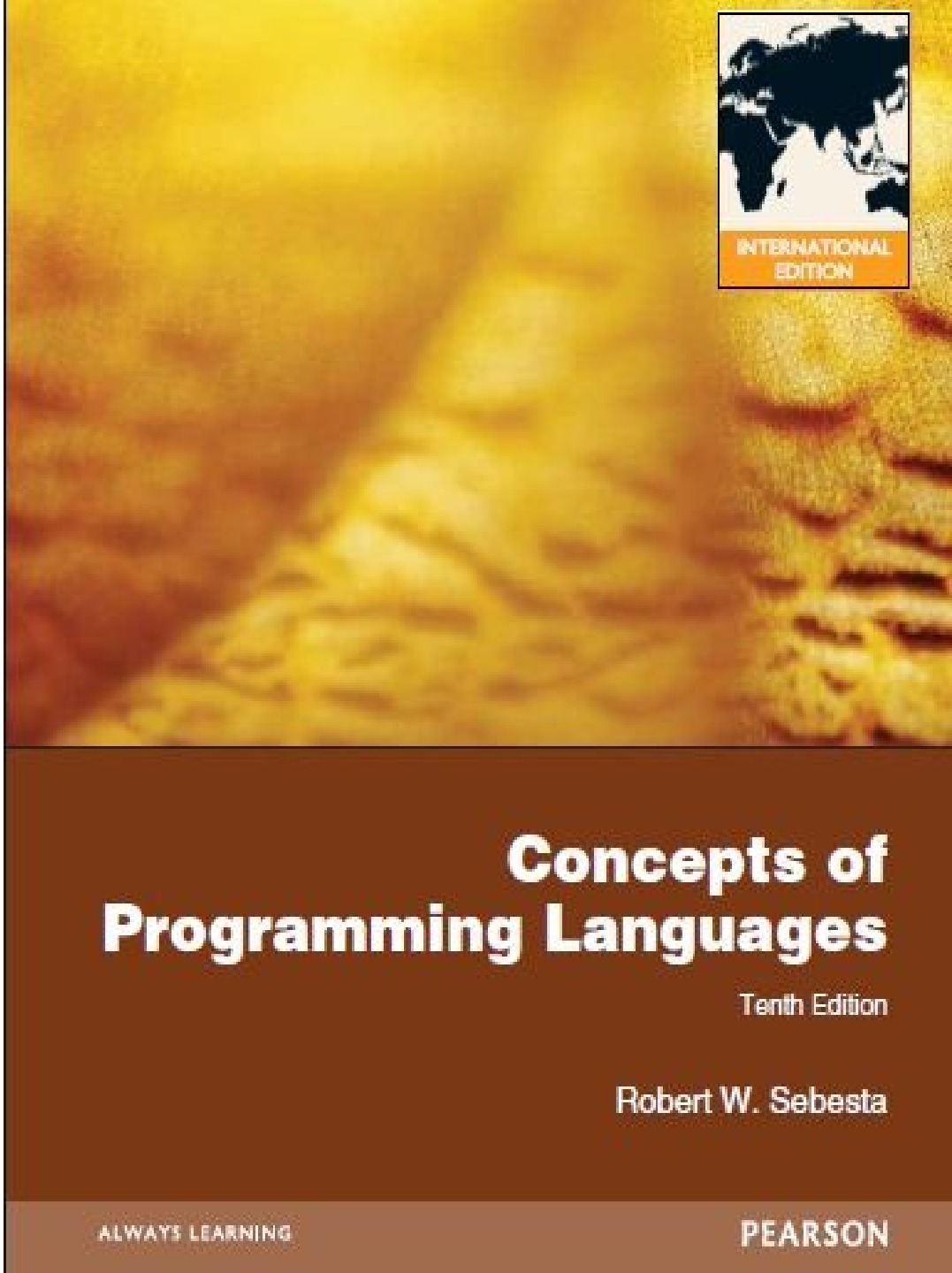
Chapter 3

Describing Syntax and Semantics



Bölüm 3 Konuları

- Giriş

Giriş

- **Sözdizimi: ifadelerin, deyimlerin ve program birimlerinin formu veya yapısı**

Sentaks (Sözdizim) ve Semantik (Anlam)

- Sözdizim ve anlam arasındaki farkı, programlama dillerinden bağımsız olarak bir örnekle incelersek:
- Tarih gg.aa.yyyy şeklinde gösteriliyor olsun.

Sözdizim	Anlam	
10.06.2007	10 Haziran 2007 6 Ekim 2007	Türkiye ABD

- Ayrıca sözdizimindeki küçük farklar anlamda büyük farklılıklara neden olabilir. Bunlara dikkat etmek gerekir:
-

```
while (i<10)          while (i<10)
{ a[i]= ++i;}         { a[i]= i++;}
```

Sözdizimini Açıklamanın Genel Sorunu: Terminoloji

- Tümce, bazı alfabeler üzerinde bir karakter dizisidir

Sözdizimini Açıklamanın Genel Sorunu: Terminoloji

- Lexeme, bir dilin en düşük düzeyli sözdizimsel birimidir (ör. *, toplam, başlangıç)

```
index = 2 * count + 17;
```

Lexemes	*Tokens*
index	identifier
=	equal_sign
2	int_literal
*	mult_op
count	identifier
+	plus_op
17	int_literal
;	semicolon

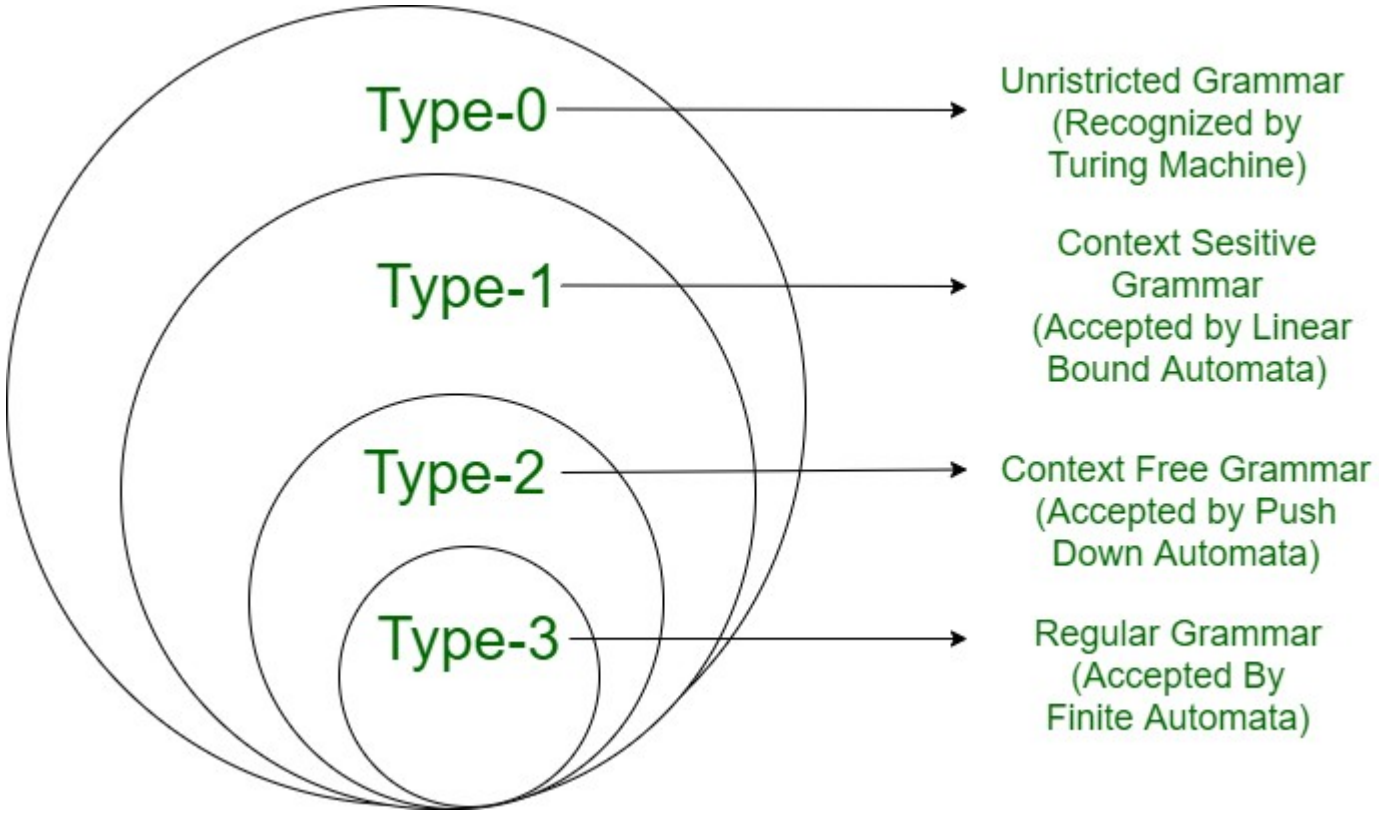
Dillerin Resmi Tanımı

- **Tanıyıcı**

BNF ve Bağlamsız Dilbilgisi

- Bağlamsız Dilbilgisi

Noam Chomsky hiyerarşisi



2.3 Context-Free Grammars

A BNF is a way of describing the grammar of a language. Most interesting grammars are context-free, meaning that the contents of any syntactic category in a sentence are not dependent on the context in which it is used. A context-free grammar is defined as a four tuple:

$$G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$$

where

- $\mathcal{N}$ is a set of symbols called nonterminals or syntactic categories.
- $\mathcal{T}$ is a set of symbols called terminals or tokens.
- $\mathcal{P}$ is a set of productions of the form $n \rightarrow \alpha$ where $n \in \mathcal{N}$ and $\alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*$.
- $\mathcal{S} \in \mathcal{N}$ is a special nonterminal called the start symbol of the grammar.

Informally, a context-free grammar is a set of nonterminals and terminals. For each nonterminal there are one or more productions with strings of zero or more nonterminals and terminals on the right hand side as described in the BNF description. There is one special nonterminal called the start symbol of the grammar.

Meta dil

A *metalanguage* is a higher-level language used to specify, discuss, describe, or analyze another language. English is used as a metalanguage for describing programming languages, but because of the ambiguities in English, more formal metalanguages have been developed. The next section describes a formal metalanguage

BNF Temelleri

-
-
-
-
-

<assign> → <var> = <expression>

<syntactic category> ::= a string of terminals and nonterminals

BNF Temelleri (devamı)

- Nonterminals genellikle açılı braketler içine alınmış

BNF Kuralları

The abstractions in a BNF description, or grammar, are often called **nonterminal symbols**, or simply **nonterminals**, and the lexemes and tokens of the rules are called **terminal symbols**, or simply **terminals**. A BNF description, or **grammar**, is a collection of rules.

Listeri Açıklama

- Sözdizimi listeleri özyineleme kullanılarak açıklanmıştır

```
| ident, <ident_list>
```

- Türetme, başlangıç sembolünden başlayıp bir cümleyle biten kuralların tekrarlanan bir uygulamasıdır (tüm terminal sembolleri)

Türetme

- Türetmedeki her sembol dizisi sentential bir formdur

Örnek Bir Dilbilgisi

<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → a | b | c | d

<expr> → <term> + <term> | <term> - <term>

<term> → <var> | const

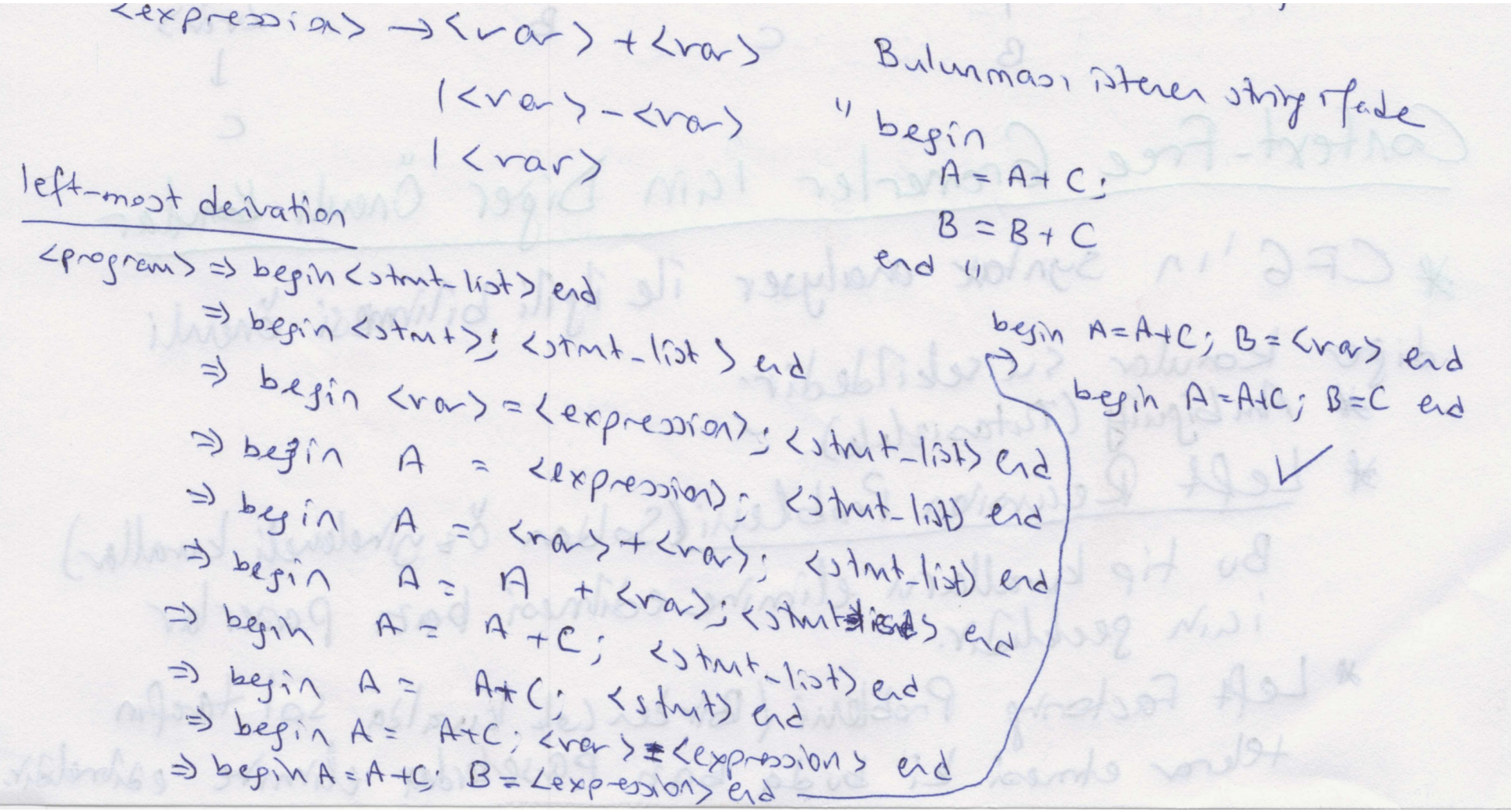
Örnek Türetme

Example : a = b + const

```
<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | const
```

<program> => <stmts> => <stmt>
=> <var> = <expr>
=> a = <expr>
=> a = <term> + <term>
=> a = <var> + <term>
=> a = b + <term>
=> a = b + const

Örnek : En Soldaki Türetme



Örnek : En Sağ Türetme

right-most derivation

<program> ⇒ begin <stmt_list> end

⇒ begin <stmt> ; <stmt_list> end

⇒ begin <stmt> ; <stmt> end

⇒ begin <stmt> ; <var> = <expression> end

⇒ begin <stmt> ; <var> = <var> end

⇒ begin <stmt> ; <var> = C end

⇒ begin <stmt> ; B = C end

⇒ begin <var> = <expression> ; B = C end

⇒ begin <var> = <var> + <var> ; B = C end

⇒ begin <var> = <var> + C ; B = C end

⇒ begin <var> = B + C ; B = C end

⇒ begin A = B + C ; B = C end ✓

Örnek : en solda

2.3.1 The Infix Expression Grammar

A context-free grammar for infix expressions can be specified as $G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ where

$\mathcal{N} = \{E, T, F\}$
$\mathcal{T} = \{identifier, number, +, -, *, /, (,)\}$
$\mathcal{P}$ is defined by the set of productions

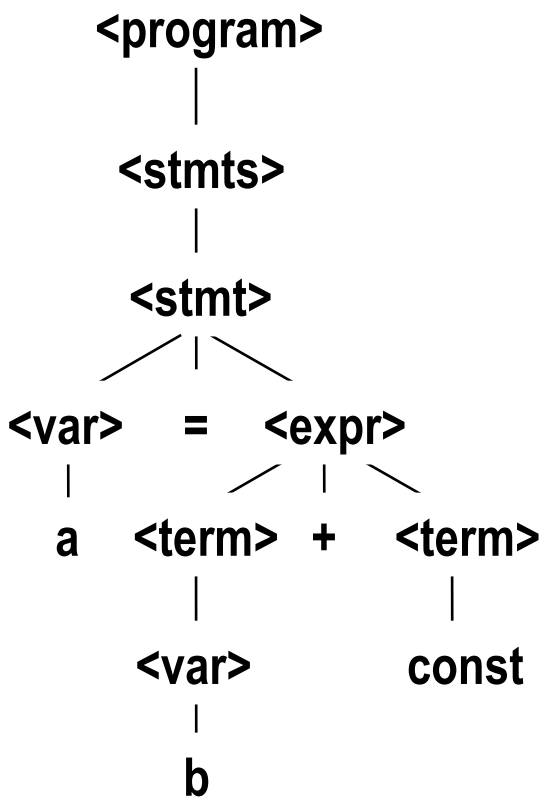
$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow (E) \mid identifier \mid number$$

$$(5 * x) + y$$

$$\underline{E} \Rightarrow \underline{E} + T \Rightarrow \underline{T} + T \Rightarrow \underline{F} + T \Rightarrow (\underline{E}) + T \Rightarrow (\underline{T}) + T \Rightarrow (\underline{T} * F) + T$$
$$\Rightarrow (\underline{F} * F) + T \Rightarrow (5 * \underline{F}) + T \Rightarrow (5 * x) + \underline{T} \Rightarrow (5 * x) + \underline{F} \Rightarrow (5 * x) + y$$

Ayrıştırma Ağacı

- Türetmenin hiyerarşik gösterimi



Ayrıştırma Ağacı

A grammar, G, can be used to build a tree representing a sentence of $L(G)$, the language of the grammar G. This kind of tree is called a *parse tree*. A parse tree is another way of representing a sentence of a given language. A parse tree is constructed with the start symbol of the grammar at the root of the tree. The children of each node in the tree must appear on the right hand side of a production with the parent on the left hand side of the same production. A program is syntactically valid if there is a parse tree for it using the given grammar.

Örnek : En Soldaki Türetme

Sola dayalı türetme

a=b*(a+c)

<assign> => <id>=<expr>

=>a=<expr>

=>a=<id>*<expr>

=>a=b*<expr>

=>a=b*(<expr>)

=>a=b*(<id>+<expr>)

=>a=b*(a+<expr>)

=>a=b*(a+<id>)

=>a=b*(a+c)

Gramer

a=b*(a+c)

<assign> → <id>=<expr>

<id> → a | b | c

<expr> → <id> + <expr>

|<id > * <expr>

|(<expr>)

| id

Örnek: En soldaki Türetme – ayrıştırma ağacı

Sola dayalı türetme

a=b*(a+c)

<assign> => <id>=<expr>

=>a=<expr>

=>a=<id>*<expr>

=>a=b*<expr>

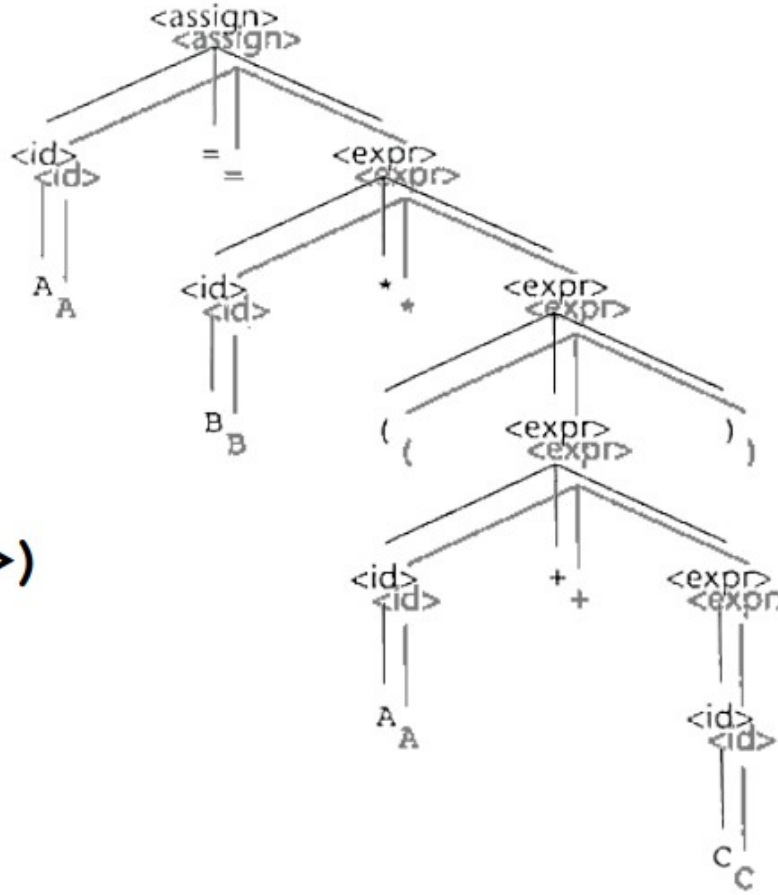
=>a=b*(<expr>)

=>a=b*(<id>+<expr>)

=>a=b*(a+<expr>)

=>a=b*(a+<id>)

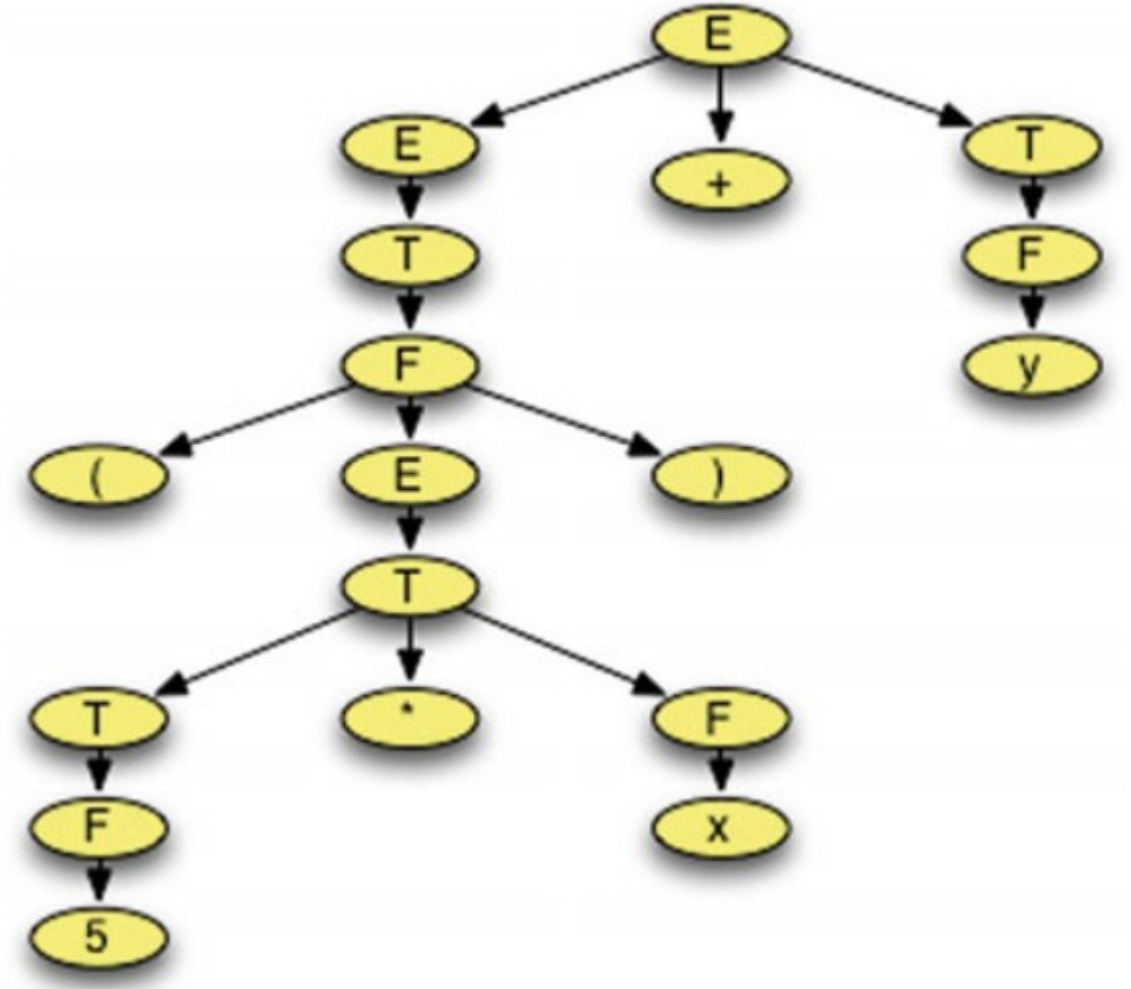
=>a=b*(a+c)



Örnek: En soldaki Türetme – ayrıştırma ağacı

$$\underline{E} \Rightarrow \underline{E} + T \Rightarrow \underline{T} + T \Rightarrow \underline{F} + T \Rightarrow (\underline{E}) + T \Rightarrow (\underline{T}) + T \Rightarrow (\underline{T} * F) + T$$
$$\Rightarrow (\underline{F} * F) + T \Rightarrow (5 * \underline{F}) + T \Rightarrow (5 * x) + \underline{T} \Rightarrow (5 * x) + \underline{F} \Rightarrow (5 * x) + y$$

$(5 * x) + y$



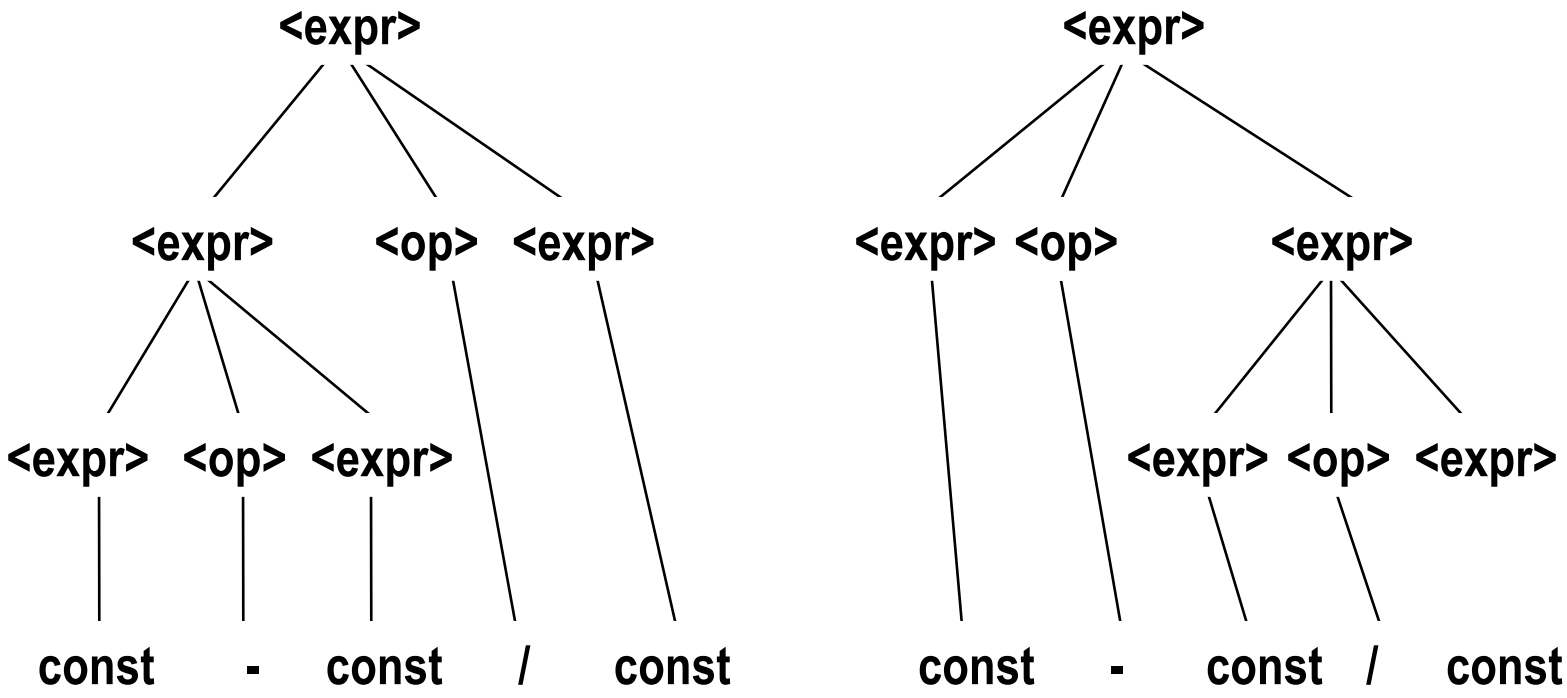
Dilbilgisinde Belirsizlik

- Dilbilgisi, yalnızca iki veya daha fazla ayrıştırıcı ağacı olan bir duyarlı form oluşturursa belirsizdir

Belirsiz İfade Dilbilgisi

<expr> → <expr> <op> <expr> | const

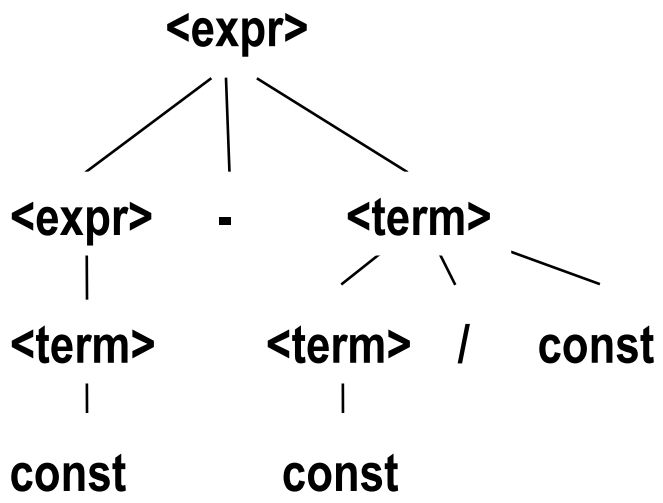
<op> → / | -



Kesin İfade Dilbilgisi

- İşleçlerin öncelik düzeylerini belirtmek için ayrıştırma ağacını kullanırsak, belirsizliğimiz olamaz

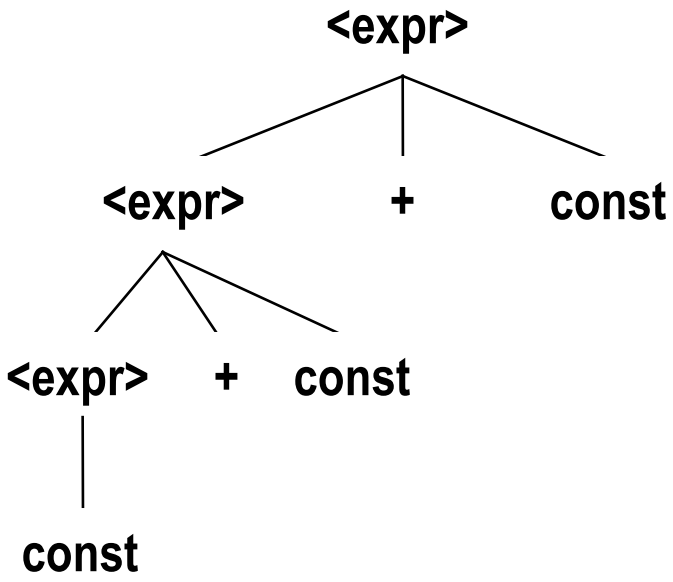
<expr> → <expr> - <term> | <term>
<term> → <term> / const| const



operatörlerin ilişkilendirilebilirliği

- İşleç ilişkilendirilebilirliği dilbilgisi ile de belirtilebilir

```
<expr> -> <expr> + <expr> |  const  (ambiguous)
<expr> -> <expr> + const  |  const  (unambiguous)
```



Genişletilmiş BNF

- İsteğe bağlı parçalar braketlere yerleştirilir [ ]
- RHS'lerin alternatif parçaları parantez içine yerleştirilir ve dikey çubuklarla ayrılır
- Tekrarlar (0 veya daha fazla) ayraçların içine yerleştirilir { }
-
-

BNF ve EBNF

- BNF

```
<expr> → <expr> + <term>
         | <expr> - <term>
         | <term>
<term> → <term> * <factor>
         | <term> / <factor>
       | <factor>
```

- EBNF

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

EBNF'deki Son Varyasyonlar

- Alternatif RHS'ler ayrı hatlara konur
- İsteğe bağlı parçalar için Kullanımı `opt`
- Seçimler için Kullanımı `one of`

Statik Anlambilim

- Anlamla ilgisi yok.

Öznitelik Dilbilgisi

- Öznitelik dilbilgisi (AG'ler), ayrıştırma ağacı düğümlerinde bazı anlamsal bilgileri taşımak için CFG'lere eklemelere sahiptir

Öznitelik Dilbilgisi : Tanım

- Def: Öznitelik dilbilgisi, aşağıdaki eklemelerle bağlamsız bir dilbilgisi G = (S, N, T, P) olur:

Öznitelik Dilbilgisi: Tanım

- Let $X_0 \rightarrow X_1 \ldots X_n$ be a rule
- Functions of the form $S(X_0) = f(A(X_1), \ldots, A(X_n))$ define *synthesized attributes*
- Functions of the form $I(X_j) = f(A(X_0), \ldots, A(X_n))$, *for* $i <= j <= n$, define *inherited attributes*
- Initially, there are *intrinsic attributes* on the leaves

Öznitelik Dilbilgisi: Bir Örnek

- Syntax

  <assign> -> <var> = <expr>

  <expr> -> <var> + <var> | <var>

  <var> A | B | C

- actual_type: synthesized for <var> and <expr>
- expected_type: inherited for <expr>

Öznitelik Dilbilgisi (devamı)

- Syntax rule: <expr> → <var>[1] + <var>[2]

  Semantic rules:

  <expr>.actual_type ← <var>[1].actual_type

  Predicate:

  <var>[1].actual_type == <var>[2].actual_type

  <expr>.expected_type == <expr>.actual_type

- Syntax rule: <var> → id

  Semantic rule:

  <var>.actual_type ← lookup (<var>.string)

Öznitelik Dilbilgisi (devamı)

- Öznitelik değerleri nasıl hesaplanır?

Attribute Grammars (continued)

<expr>.expected_type ← inherited from parent

<var>[1].actual_type ← lookup (A)
<var>[2].actual_type ← lookup (B)
<var>[1].actual_type =? <var>[2].actual_type

<expr>.actual_type ← <var>[1].actual_type
<expr>.actual_type =? <expr>.expected_type

Anlambilim

- Anlambilimi tanımlamak için yaygın olarak kabul edilebilir tek bir notasyon veya formalizm yoktur

Operasyonel Anlambilim

- Operasyonel Anlambilim

Operasyonel Anlambilim

- Bir donanım saf tercüman çok pahalı olurdu

Operasyonel Anlambilim (devamı)

- Daha iyi bir alternatif: Eksiksiz bir bilgisayar simülasyon

Operasyonel Anlambilim (devamı)

- Operasyonel anlambilimin kullanımları:

Denotasyonal Anlambilim

- Özyinelemeli fonksiyon teorisine dayalı

Denotasyonal Anlambilim – devamı

- Bir dil için denotasyon belirtimi oluşturma işlemi:

Denotasyonal Anlambilim: program durumu

- Bir programın durumu, tüm geçerli değişkenlerinin değerleridir
- VARMAP, değişken adı ve durumu verildiğinde değişkenin geçerli değerini döndüren bir işlev olsun

Ondalık Sayılar

```
<dec_num> → '0' | '1' | '2' | '3' | '4' | '5' |
            '6' | '7' | '8' | '9' |
            <dec_num> ('0' | '1' | '2' | '3' |
                       '4' | '5' | '6' | '7' |
                       '8' | '9')
```

M_{dec}('0') = 0, M_{dec} ('1') = 1, …, M_{dec} ('9') = 9

M_{dec} (<dec_num> '0') = 10 * M_{dec} (<dec_num>)

M_{dec} (<dec_num> '1') = 10 * M_{dec} (<dec_num>) + 1

…

M_{dec} (<dec_num> '9') = 10 * M_{dec} (<dec_num>) + 9

İfade

- İfadeleri haritala Z ∪ {error}
- İfadelerin ondalık sayılar, değişkenler veya ikili ifadeler olduğunu varsayıyoruz, her biri bir ifade olabilen bir aritmetik işleç ve iki işlenene sahip
-

İfade

```
Me(<expr>, s) Δ=
    case <expr> of
      <dec_num> => Mdec(<dec_num>, s)
      <var> =>
           if VARMAP(<var>, s) == undef
                then error
                else VARMAP(<var>, s)
     <binary_expr> =>
          if (Me(<binary_expr>.<left_expr>, s) == undef
                OR Me(<binary_expr>.<right_expr>, s) =
                                 undef)
              then error
           else
           if (<binary_expr>.<operator> == '+' then
              Me(<binary_expr>.<left_expr>, s) +
                     Me(<binary_expr>.<right_expr>, s)
           else Me(<binary_expr>.<left_expr>, s) *
               Me(<binary_expr>.<right_expr>, s)
        ...
```

Atama Deyimleri

- Durum kümelerini durum kümelerine eşler U {error}

M_a(x := E, s) Δ=

if M_e(E, s) == error

then error

else s′ =

{<i_1,v_1′>,<i_2,v_2′>,...,<i_n,v_n′>},

where for j = 1, 2, ..., n,

if i_j == x

then v_j′ = M_e(E, s)

else v_j′ = VARMAP(i_j, s)

Mantıksal Ön Test Döngüleri

- Durum kümelerini durum kümelerine eşler U {error}

M_l(while B do L, s) Δ=
    if M_b(B, s) == undef
        then error
        else if M_b(B, s) == false
            then s
            else if M_{sl}(L, s) == error
                then error
                else M_l(while B do L, M_{sl}(L, s))

Döngü Anlamı

- Döngünün anlamı, döngüdeki ifadeler, herhangi bir hata olmadığı varsayılarak, öngörülen sayıda çalıştırıldıktan sonra program değişkenlerinin değeridir.

Denotasyonel Anlambilimin Değerlendirilmesi

- Programların doğruluğunu kanıtlamak için kullanılabilir

Aksiyomatik Anlambilim

- Resmi mantığa dayalı (yüklem hesabı)

Aksiyomatik Anlambilim (devamı)

- Bir deyimden önceki bir onaylama işlemi (ön koşul), yürütmenin o noktasında doğru olan değişkenler arasındaki ilişkileri ve kısıtlamaları belirtir

Aksiyomatik Anlambilim Formu

- Ön, posta formu: `{P} statement {Q}`

- Bir örnek
  - Olası bir ön koşul: `{b > 10}`
  - En zayıf ön koşul: `{b > 0}`

Program Prova süreci

- Tüm program için postcondition istenen sonuçtur

Aksiyomatik Anlambilim: Atama

- Atama deyimleri için bir aksiyom
  (x = E): $\{Q_{x->E}\}$ x = E {Q}

- Sonucun Kuralı:

$$\frac{\{P\}\ S\ \{Q\},\ P' \Rightarrow P,\ Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

Aksiyomatik Anlambilim: Diziler

- Formun dizileri için çıkarım kuralı S1; S2

{P1} S1 {P2}
{P2} S2 {P3}

$$\frac{\{P1\}\ S1\ \{P2\},\ \{P2\}\ S2\ \{P3\}}{\{P1\}\ S1; S2\ \{P3\}}$$

Aksiyomatik Anlambilim: Seçim

- Seçim için çıkarım kuralları

$$\frac{\{B \text{ and } P\}\ S1\ \{Q\},\ \{(\text{not } B) \text{ and } P\}\ S2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ S1\ \texttt{else}\ S2\ \{Q\}}$$

Aksiyomatik Anlambilim: Döngüler

- Mantıksal ön test döngüleri için çıkarım kuralı

  {P} **`while`** B **`do`** S **`end`** {Q}

$$\frac{\{\text{I and B}\}\ \text{S}\ \{I\}}{\{I\}\ \text{while B do S}\ \{\text{I and (not B)}\}}$$

döngü değişmez olduğum yerde (endüktif hipotez)

Aksiyomatik Anlambilim: Aksiyomlar

- Döngünün değişmez özellikleri: Aşağıdaki koşulları karşılamalıyım:

Döngü Değişmez

- Döngü değişmez I, döngü postcondition'ının zayıflamış bir sürümüdür ve aynı zamanda bir ön koşuldur.

Evaluation of Axiomatic Semantics

- Developing axioms or inference rules for all of the statements in a language is difficult
- It is a good tool for correctness proofs, and an excellent framework for reasoning about programs, but it is not as useful for language users and compiler writers
- Its usefulness in describing the meaning of a programming language is limited for language users or compiler writers

Açıklama Semantiği vs Operasyonel Anlambilim

- Operasyonel anlambilimde, durum değişiklikleri kodlanmış algoritmalar tarafından tanımlanır

Özet

- BNF ve bağlam içermeyen dilbilgisi eşdeğer meta dillerdir