


Bölümün Amacı

- Programlama dillerinde sözdizim ve anlam kavramları,
- Soyut sözdizim, metinsel sözdizim ve gramer kavramları,
- Ayırıştırma ağaçlarının ve türetmelerin oluşturulması,
- Sözdizim tanımlanması için kullanılan BNF metadili,
- Anlam tanımlamada ve dilin standartlaştırılması

GİRİŞ

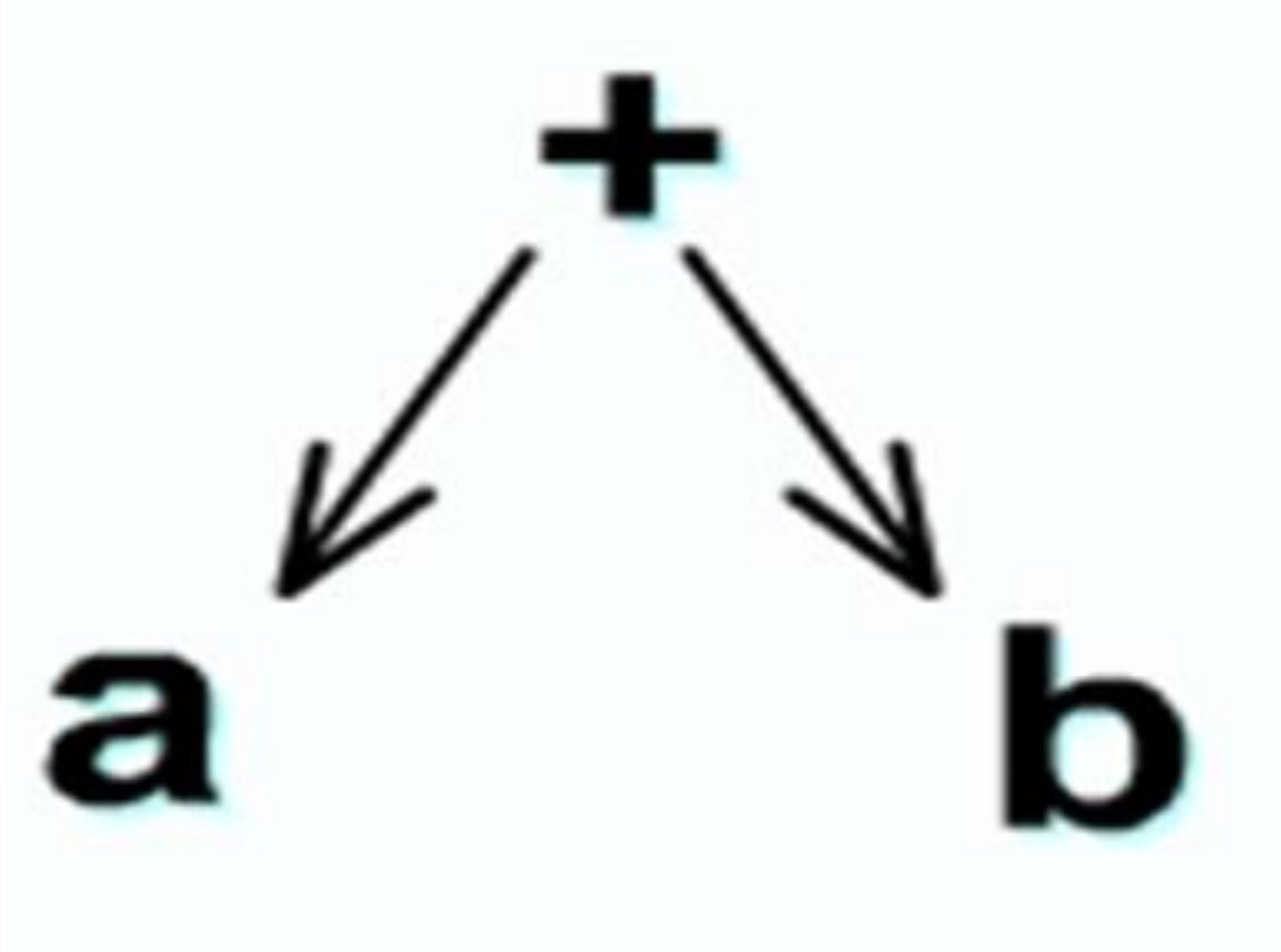
- Yazılan bir programın o dile ait olup olmadığını belirleyen kurallar, **sözdizim** (*syntax*) ve **anlam** (*semantics*) olarak ikiye ayrılabilir.
- Bir programın **begin** ile başlayıp **end** ile bitmesi, her deyimin sonunda noktalı virgül bulunması (sözdizim kuralları), bir değişkenin kullanılmadan önce tanımlanması (anlam kuralı) gibi.

- 
- Bir dilin sözdizimini anlatmak amacıyla kullanılan bir araç vardır. BNF (Backus-Naur Form) **metadili** böyle bir araçtır.
 - Anlam tanımlama için böyle bir dil yoktur.

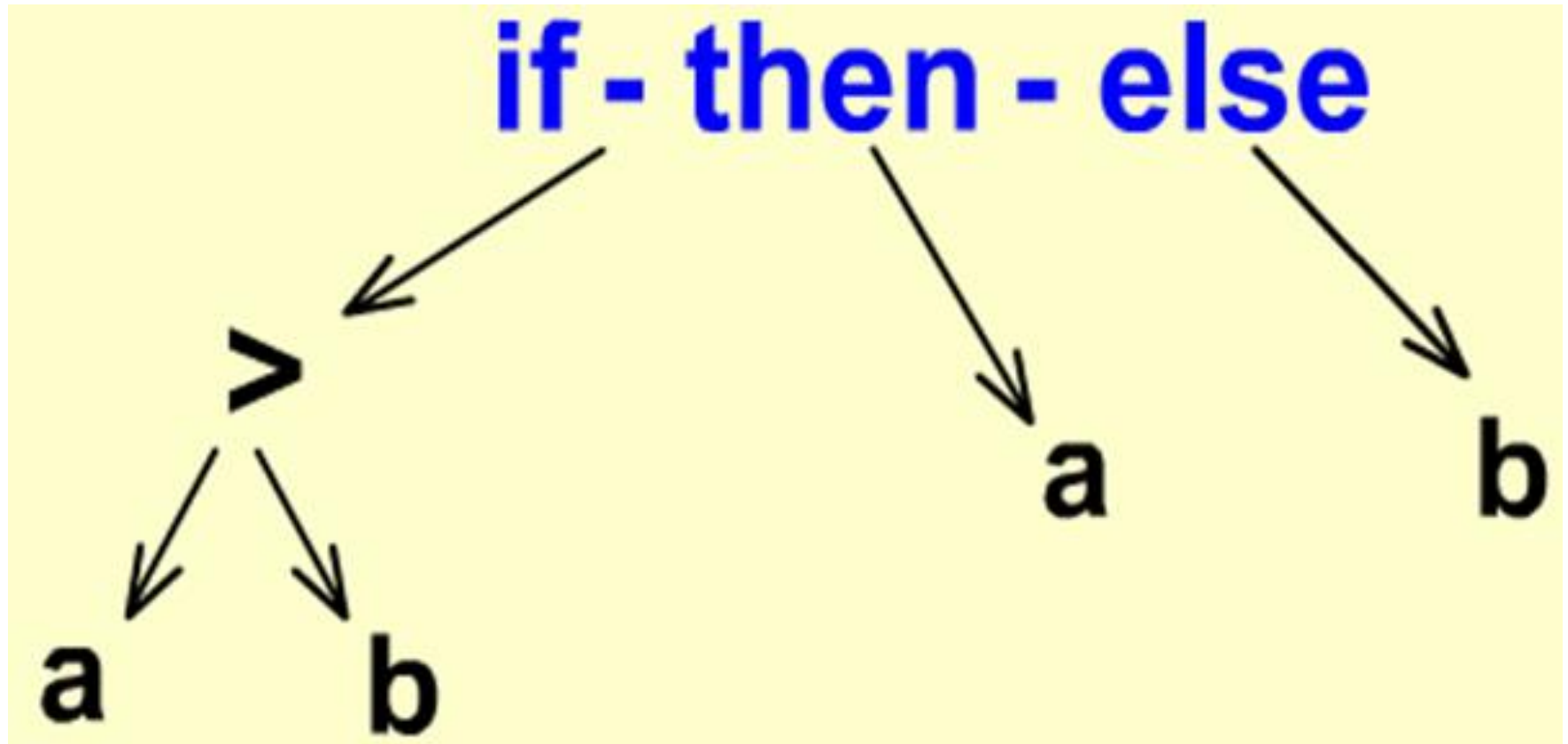
SOYUT SÖZDİZİM

- Bir dilde bulunan her yapıdaki anlamlı bileşenler o dilin **soyut sözdizimi**, tanımlar.
- $+ab$ *prefix* ifadesi,
- $a+b$ *infix* ifadesi,
- $ab+$ *postfix* ifadelerindeki $+$ işlemcisi ve a ve b alt-ifadelerinden oluşan aynı anlamlı bileşenleri içermektedir.

Bu nedenle ağaç olarak üçünün de gösterimi yandaki şekilde gibidir.



if $a > b$ then a else b



METİNSEL SÖZDİZİM

- Dil, bir alfabedeki karakter dizilerinden oluşurlar.
- Dildeki karakter dizileri **cümle** veya **deyim** oluşturur.
- **Sözdizim kuralları:** Hangi karakter dizilerinin o dilin alfabesinden ve o dile ait olduğunu belirlerler.
- Diller sözdizimsel olarak basittir.

SÖZDİZİM (devam)

- **Lexeme**: Bir programlama dilindeki en düşük düzeyli sözdizimsel birimlerdir.
- Programlar, karakterler yerine *lexeme*'ler dizisi olarak düşünülebilir.
- **Token** : Bir dildeki *lexeme*'lerin gruplanması oluşan sınıflardır.

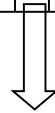
- Dilinin metinsel sözdizimi, *token*'lar dizisidir.
- Örneğin bir tanımlayıcı (identifier); *ortalama* veya *ogrenci* gibi *lexeme*'leri olabilen bir *token*'dır.
- Bir *token*'ın sadece tek bir olası *lexeme*'i olabilir.

Örneğin, çarpma_ışlemcisi denilen aritmetik işlemci "*" sembolü için, tek bir olası *lexeme* vardır.

- Boşluk (*space*), ara (*tab*) veya yeni satır karakterleri, *token*'lar arasına yerleştirilmesi programın anlamına etkisi yoktur.

character stream

v a l = 1 0 * v a l + i



lexical analysis (scanning)



token stream

1	3	2	4	1	5	1
(ident)	(assign)	(number)	(times)	(ident)	(plus)	(ident)
"val"	-	10	-	"val"	-	"i"

token number

token value

PROGRAMLAMA DİLLERİNDE GRAMER

- Bir programlama dilinin metinsel (somut) sözdizimini açıklamak için **Gramer**, kullanılır.
- Gramerler, anahtar kelimelerin (keywords) ve noktalama işaretlerinin yerleri; deyim listelerinin nasıl oluşturulacağını belirleyen bir dizi kuraldan oluşur.

character stream

v a l = 1 0 * v a l + i



lexical analysis (scanning)



token stream

1	3	2	4	1	5	1
(ident)	(assign)	(number)	(times)	(ident)	(plus)	(ident)
"val"	-	10	-	"val"	-	"i"

token number

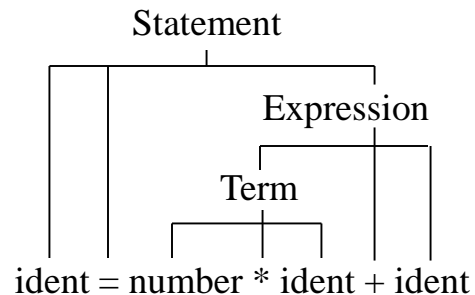
token value



syntax analysis (parsing)



syntax tree



Grammar Nedir?

Example

Statement = "if" "(" Condition ")" Statement ["else" Statement].

4 bileşenden oluşur

terminal symbols

atomik

"if", ">=", ident, number, ...

nonterminal symbols

Sözdizim değişkenleri

Statement, Expr, Type, ...

productions

Nonterminallerin çözümü

Statement = Designator "=" Expr ";" .
Designator = ident ["." ident].

...

start symbol

Başlangıç nonterminali

begin

Arithmetik İfadelerin Grameri

Productions

$Expr = ["+" | "-"] Term \{ ("+" | "-") Term \}.$
 $Term = Factor \{ ("*" | "/") Factor \}.$
 $Factor = ident | number | "(" Expr ")".$

Terminal symbols

simple TS: $+, -, *, /, (,)$
(just 1 instance)

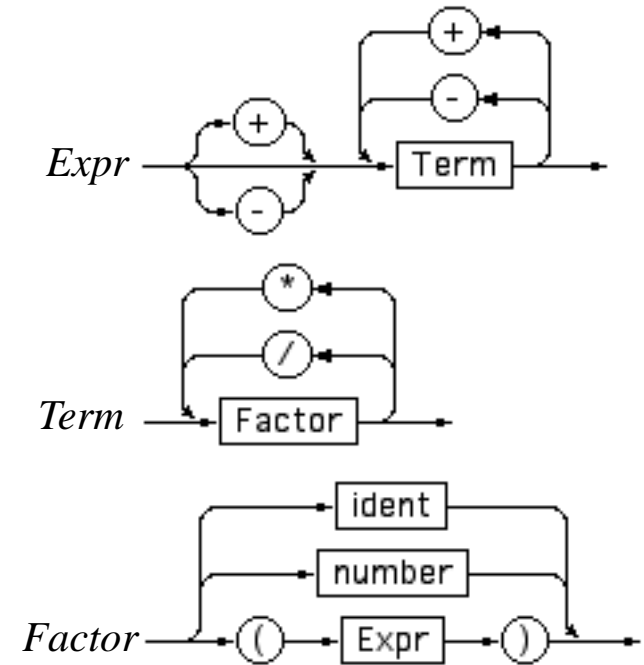
terminal classes: $ident, number$
(multiple instances)

Nonterminal symbols

$Expr, Term, Factor$

Start symbol

$Expr$



Backus-Naur Form (BNF)-CFG

□ CFG

$$\begin{aligned} \text{expression} \rightarrow & \text{identifier} \mid \text{number} \mid - \text{expression} \\ & \mid (\text{expression}) \\ & \mid \text{expression operator expression} \end{aligned}$$
$$\text{operator} \rightarrow + \mid - \mid * \mid /$$

□ BNF

$$\begin{aligned} \langle \text{expression} \rangle \rightarrow & \langle \text{identifier} \rangle \mid \langle \text{number} \rangle \mid - \\ & \langle \text{expression} \rangle \\ & \mid (\langle \text{expression} \rangle) \\ & \mid \langle \text{expression} \rangle \langle \text{operator} \rangle \langle \text{expression} \rangle \end{aligned}$$
$$\langle \text{operator} \rangle \rightarrow + \mid - \mid * \mid /$$

EBNF Notation

Extended Backus-Naur form

John Backus: developed the first Fortran compiler

Peter Naur: edited the Algol60 report

<i>symbol</i>	<i>meaning</i>	<i>examples</i>
string		"=", "while"
name		ident, Statement
=		A = b c d .
.		
	separates alternatives	a b c ⌚ a or b or c
(...)	groups alternatives	a (b c) ⌚ ab ac
[...]	optional part	[a] b ⌚ ab b
{...}	repetitive part	{ a } b ⌚ b ab aab aaab ...

EBNF

`<seçimlik_deyim> -> lf (<mantıksal>) <deyim> [else <deyim>];`

BNF

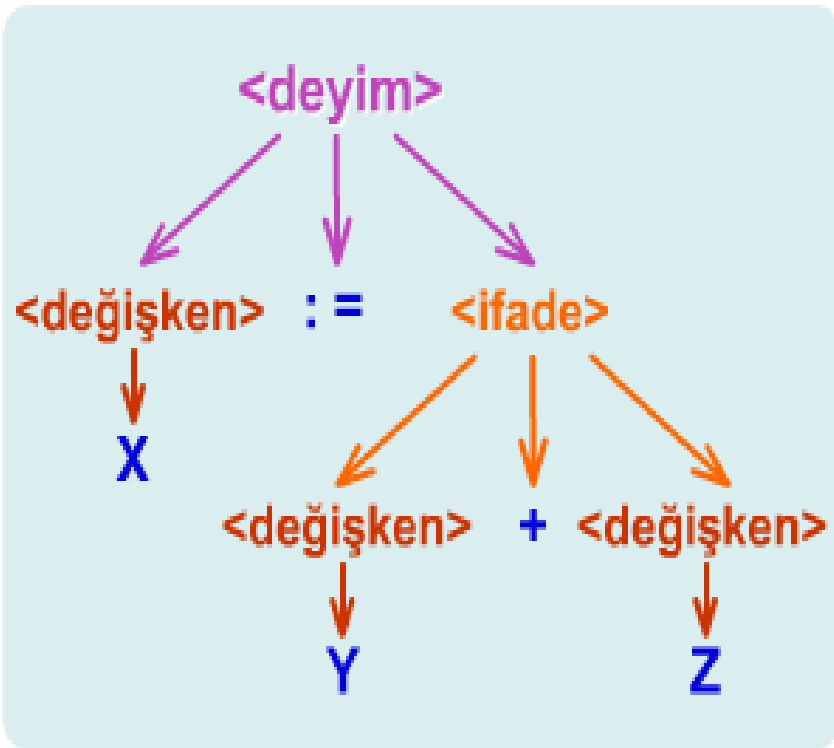
`<seçimlik_deyim> -> lf (<mantıksal>) <deyim>`

`<seçimlik_deyim> -> lf (<mantıksal>) <deyim> else <deyim>;`

Değiştirme (*alternation*) |

EBNF	<code><for_deyimi>->for<değişken> := <ifade> (to down to) <ifade> do <deyim></code>
BNF	<code><for_deyimi>->for<değişken> := <ifade> (to) <ifade> do <deyim></code>
	<code><for_deyimi>->for<değişken> := <ifade> (down to) <ifade> do <deyim></code>

Grammerler ve Türetimler



`<program> -> begin <deyim_listesi> end`

`<deyim_listesi> -> <deyim>
| <deyim>;<deyim_listesi>`

`<deyim> -> <değişken> :=<ifade>`

`<ifade> -> <değişken> + <değişken>
| <değişken>`

`<değişken> -> X | Y | Z`

Örnek

- $expression \rightarrow identifier \mid number \mid - expression$
 $\mid (expression)$
 $\mid expression operator expression$
 $operator \rightarrow + \mid - \mid * \mid /$

slope * x + intercept ifadesini
türetelim.

slope * x + intercept

expression \Rightarrow *expression operator expression*

\Rightarrow *expression operator intercept*

\Rightarrow *expression + intercept*

\Rightarrow *expression operator expression +
intercept*

\Rightarrow *expression operator x + intercept*

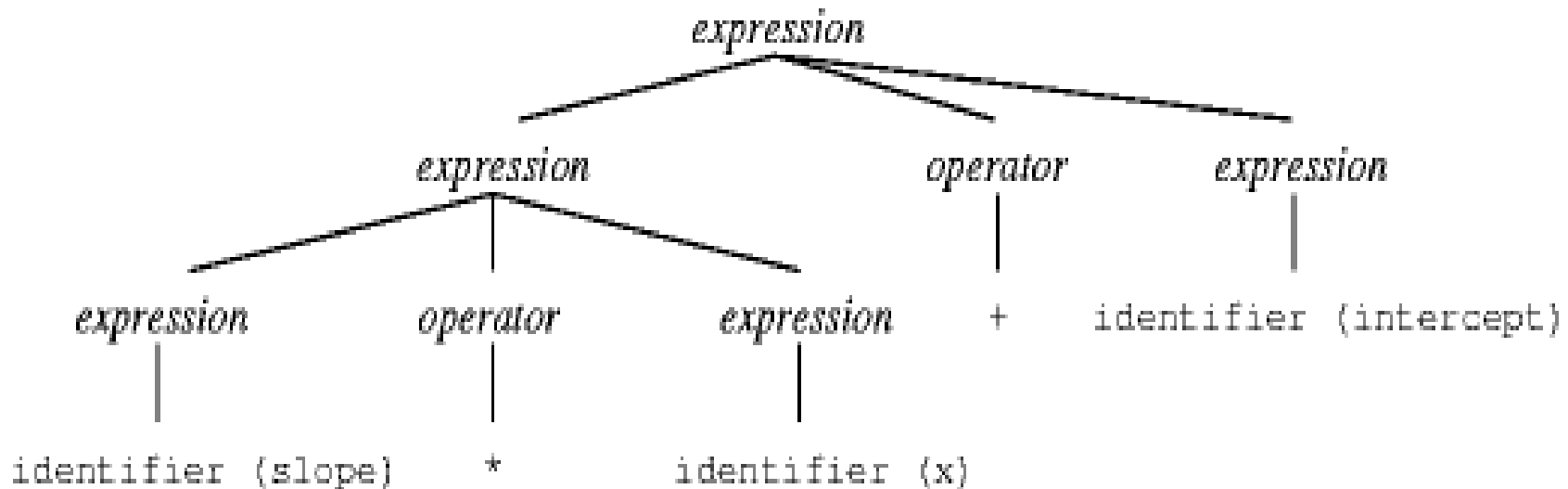
\Rightarrow *expression * x + intercept*

\Rightarrow slope * x + intercept

expression \Rightarrow^* slope * x + intercept

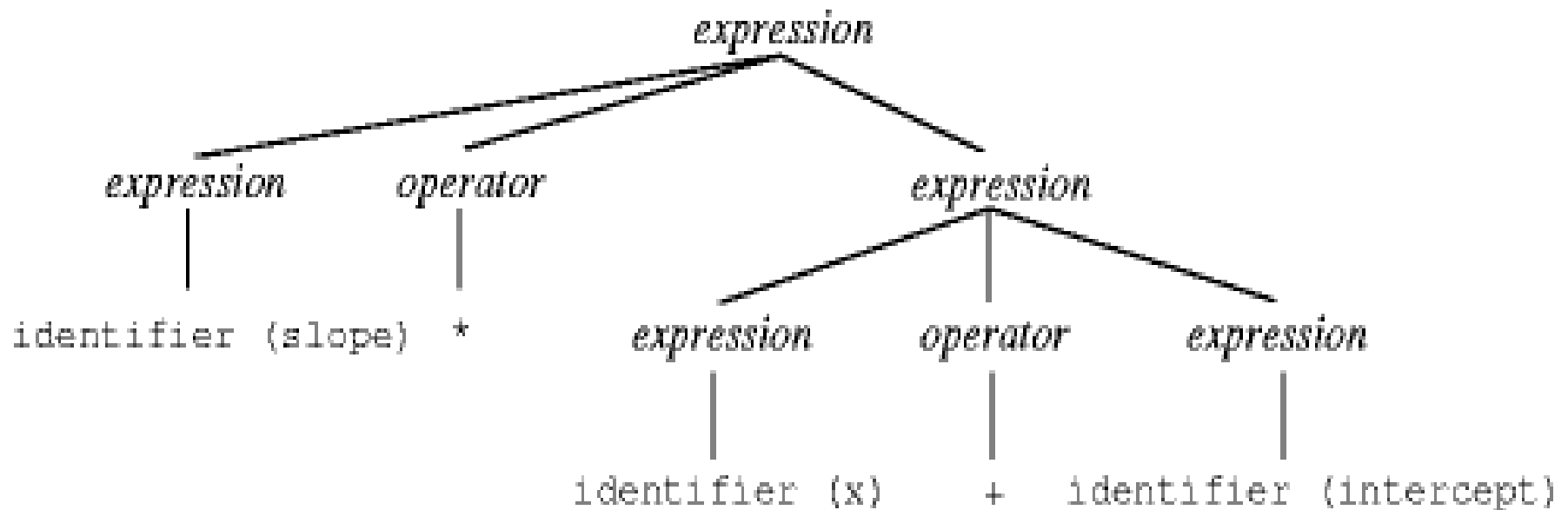
Ayrıştırma Ağacı(Parse Trees)

- Türetimin grafik gösterimi



Belirsiz Gramer

- Bir gramerde aynı ifade için alternatif ayrıştırma ağacı bulunuyorsa bu gramer belirsizdir (ambiguous)



Belirsizlik kaldırılmalıdır.

$$10 - 4 - 3 \equiv (10 - 4) - 3$$

$$3 + 4 * 5 \equiv 3 + (4 * 5)$$

$$(1) \text{ expression } \longrightarrow \text{ term } \mid \text{ expression add_op term}$$

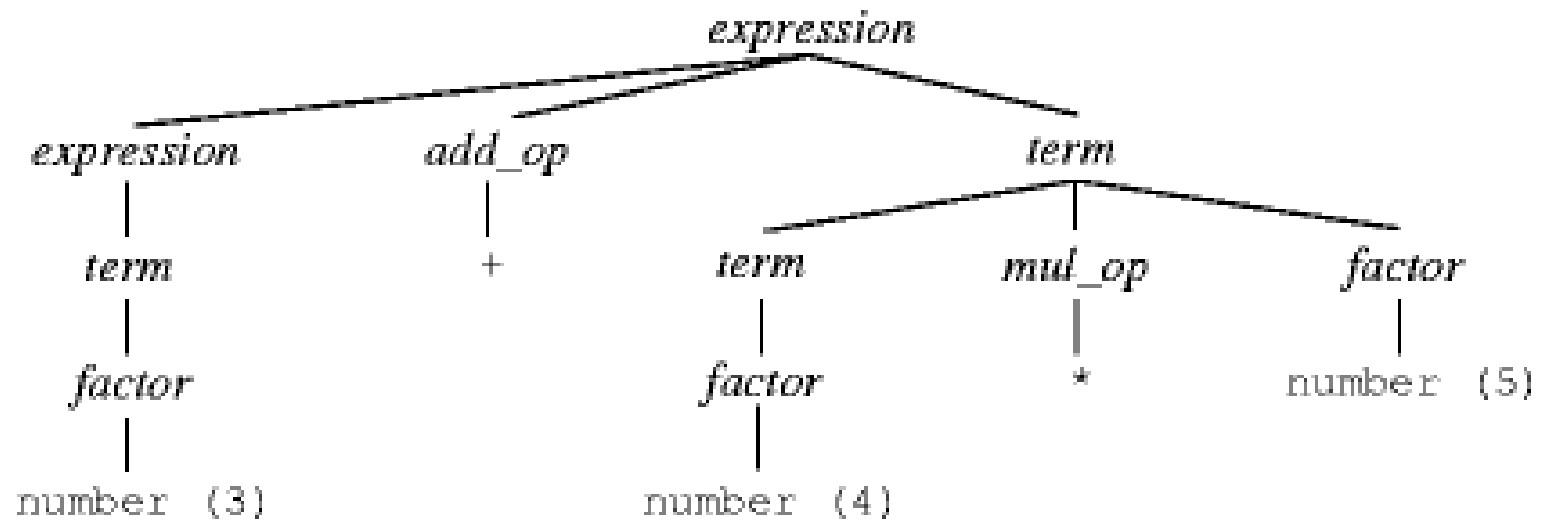
$$(2) \text{ term } \longrightarrow \text{ factor } \mid \text{ term mult_op factor}$$

$$(3) \text{ factor } \longrightarrow \text{ identifier } \mid \text{ number } \mid - \text{ factor } \mid (\text{ expression })$$

$$(4) \text{ add_op } \longrightarrow + \mid -$$

$$(5) \text{ mult_op } \longrightarrow * \mid /$$

3 + 4 * 5 için Parse ağacı



Operator Priority

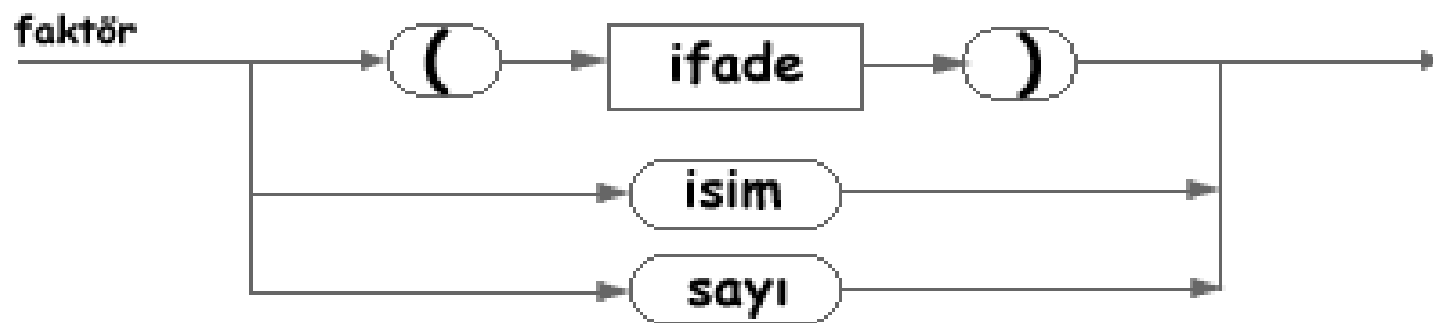
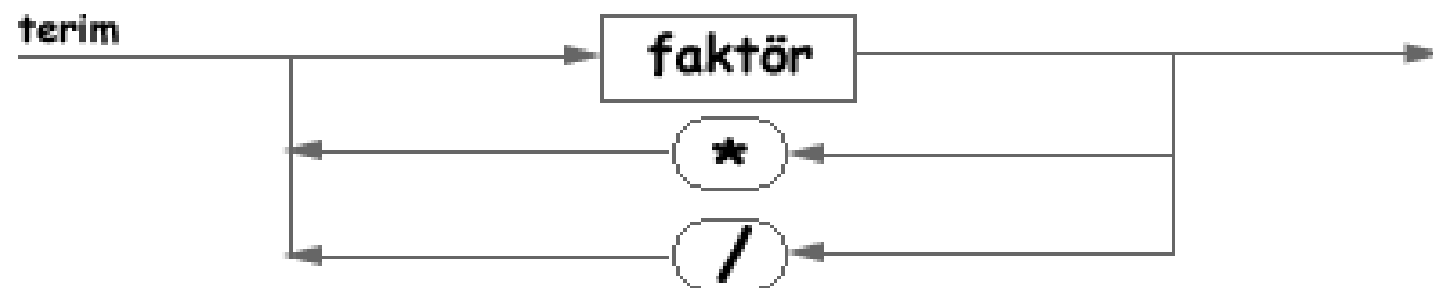
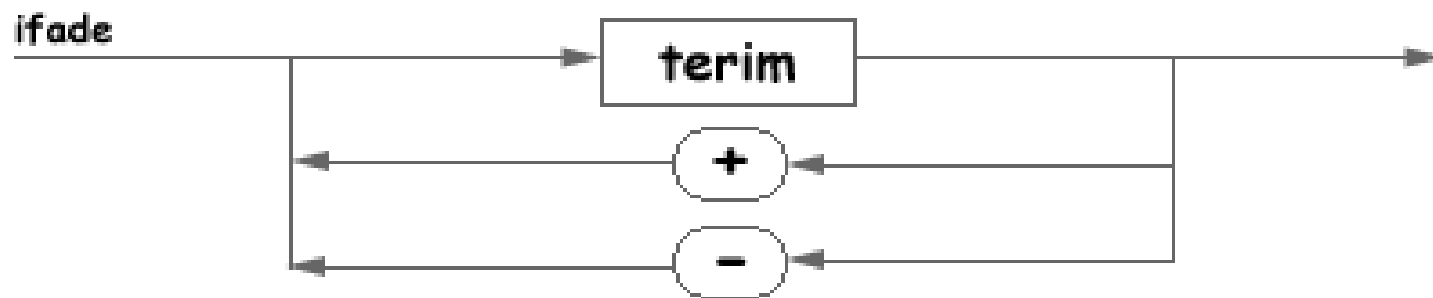
Expr = ["+" | "-"] Term { ("+" | "-") Term }.
Term = Factor { ("*" | "/") Factor }.
Factor = ident | number | "(" Expr ")".

input: - a * 3 + b / 4 - c

- ⌞ - ident * number + ident / number - ident
- ⌞ - Factor * Factor + Factor / Factor - Factor

Sözdizim Grafikleri

- BNF ve EBNF'teki kurallar, ayrıştırma ağacı dışında **sözdizim grafikleri** (*syntax graphs*) ile de gösterilebilir. Sözdizim grafikleri, ilk olarak Pascal'ın gramerini açıklamak için kullanılmıştır.



ANLAMSAAL TANIMLANMA

- Bir programlama dilinin **anlam** (*semantic*) **kuralları**, bir dilde sözdizimsel olarak geçerli olan herhangi bir programın anlamını belirler.

Anlam tanımlama

- Anlamsal tanımlama için varolan yöntemler oldukça karmaşıktır ve hiçbir yöntem sözdizim tanımlamak için kullanılan BNF metadili gibi yaygın kullanıma ulaşmamıştır.

Anlam tanımlama

- Anlam tanımlama için, dil yapılarının Türkçe gibi bir doğal dilde açıklanması sağlanır. Ancak doğal dil kullanılarak yapılan açıklamalar, açık ve kesin olmaz.



Durağan Anlam Kuralları

Dinamik anlam Kuralları

PROGRAMLAMA DİLLERİNİN STANDARTLAŞTIRILMASI

- Dilin sözdizimi ve anlamı tam ve açık olarak tanımlanmalı ve **dil standardı** oluşturulmalıdır.
- Bir dil standardı, dil tasarımcısı, dil tasarımını destekleyen kuruluş veya Amerika Birleşik Devletleri'nde *American National Standards Institute (ANSI)* yada *International Standards Organization (ISO)* tarafından gerçekleştirilebilir gibi bir kuruluş tarafından tanımlanabilir.

Bölüm Özeti

Metinsel Sözdizim
Gramer, BNF, EBNF
Türetim ve türetme ağaçları
Belirsizlik
Belirsizliğin kaldırılması

Kaynaklar:

- **Programmng Languages: Concepts&Constructs, Second Edition, Ravi Sethi.**
- **Concept og Programming Languages, Fourth Edition, Robert W. Sebesta.**