190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 11

Dr. Öğr. Üyesi İsmail Hakkı Parlak

Kaynak: METU CENG 242 Ders Notları, Şehitoğlu, O. T.

Syntax

- Söz dizimi (Syntax): Bir programlama dilindeki sembollerin, noktalama işaretlerinin ve kelimelerin yapısını kontrol eden kurallar. Şekilsel kurallar.
- Anlambilim (Semantics): Programın anlamı.
- Cümle: Bir alfabe üzerinde tanımlı karakterler dizesi.
- Dil: Cümleler kümesi.

Syntax

- Sözcükbirim (Lexeme): Dildeki en alt düzey söz dizim birimi. Örn: {, ;, +, float, my_var
- Simge (Token): Sözcükbirim kategorileri. Örn: Tanımlayıcı (identifier), operatör, anahtar sözcük (keyword), sabit (constant)

Syntax

- Sözdizim doğrulama (Syntax recognition): Bir string'in bir dilde olup olamayacağına karar verme.
- Sözdizim üretme (Syntax generation): Dil kurallarına uygun string üretme.
- Derleyici (compiler) ve yorumlayıcılar (interpreter) syntax'ı algılayıp metinleri makinenin anlayacağı forma dönüştürürler.

İçerikten Bağımsız Gramer

- İçerikten bağımsız gramer (context free grammar - CFG), belirli bir biçimsel dilde olası tüm dizileri oluşturmak için kullanılan biçimsel bir dilbilgisidir.
- CFG bir 4'lü ile şu şekilde tanımlanabilir:
- $G = (N, \Sigma, R, S)$
  - N, sonlanmayan (non terminals) elemanlar kümesi (değişkenler).
  - Σ, sonlandırıcı (terminals) elemanlar kümesi.
  - R, kurallar (rules) kümesi: $N \rightarrow (N \cup \Sigma)^*$
  - S, başlangıç sembolü. $S \in N$
- $L(G) = \{w \mid w \in \Sigma^*, S \rightarrow^* w\}$ (İçerikten bağımsız dil - CFL)

İçerikten Bağımsız Gramer

- $L_1 = \{a^n b^n \mid n >= 0\}$

  $= \{\varepsilon, ab, aabb, aaabbb, \ldots\}$
- $G_1 = (N, \Sigma, R, S)$
  - $N = \{S\}$
  - $\Sigma = \{a, b\}$
  - $R = \{S \rightarrow aSb \mid \varepsilon\}$
- "aaabbb" L_1'de bulunabilir mi?

  $S \rightarrow aSb, aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaa\varepsilon bbb \rightarrow aaabbb$

İçerikten Bağımsız Gramer

- Öz yineli (recursive) veya liste benzeri yapılar öz yineleme (recursion) ile gösterilebilir.
- Türetim (derivation): Bir değişken (non terminal) ile başlar ve kurallar elde sadece sonlandırıcı (terminal) elemanlar kalana kadar uygulanır.

İçerikten Bağımsız Gramer

- $L(G) = \{ww^R \mid w \in \{a, b\}^*\}$.
- ör: aabbaa, baab, …
- $G = \{\{S\}, \{a, b\}, R, S\}$.
- R = {

  $S \rightarrow aSa$,

  $S \rightarrow bSb$,

  $S \rightarrow \varepsilon$

  }.

İçerikten Bağımsız Gramer

- L(G) = Doğru formlu parantezler
- ör: (), (()), ()(())()((())))
- G = {{S}, {(,)}, R, S}.
- R = {
  S → SS,
  S → (S),
  S →()
  }.

İçerikten Bağımsız Gramer

⟨stmt⟩ → ⟨id⟩ = ⟨expr⟩

⟨expr⟩ → ⟨expr⟩ ⟨op⟩ ⟨expr⟩ | ⟨id⟩

⟨op⟩ → + | *

⟨id⟩ → a | b | c

Soldan Türetim (Leftmost derivation):

a = a * b

⟨stmt⟩ → ⟨id⟩ = ⟨expr⟩ → a = ⟨expr⟩ → a = ⟨expr⟩ ⟨op⟩ ⟨expr⟩ → a = ⟨id⟩ ⟨op⟩ ⟨expr⟩

→ a = a ⟨op⟩ ⟨expr⟩ → a = a * ⟨expr⟩ → a = a * ⟨id⟩ → a = a * b

İçerikten Bağımsız Gramer

⟨stmt⟩ → ⟨id⟩ = ⟨expr⟩

⟨expr⟩ → ⟨expr⟩ ⟨op⟩ ⟨expr⟩ | ⟨id⟩

⟨op⟩ → + | *

⟨id⟩ → a | b | c

Sağdan Türetim (Rightmost derivation):

a = a * b

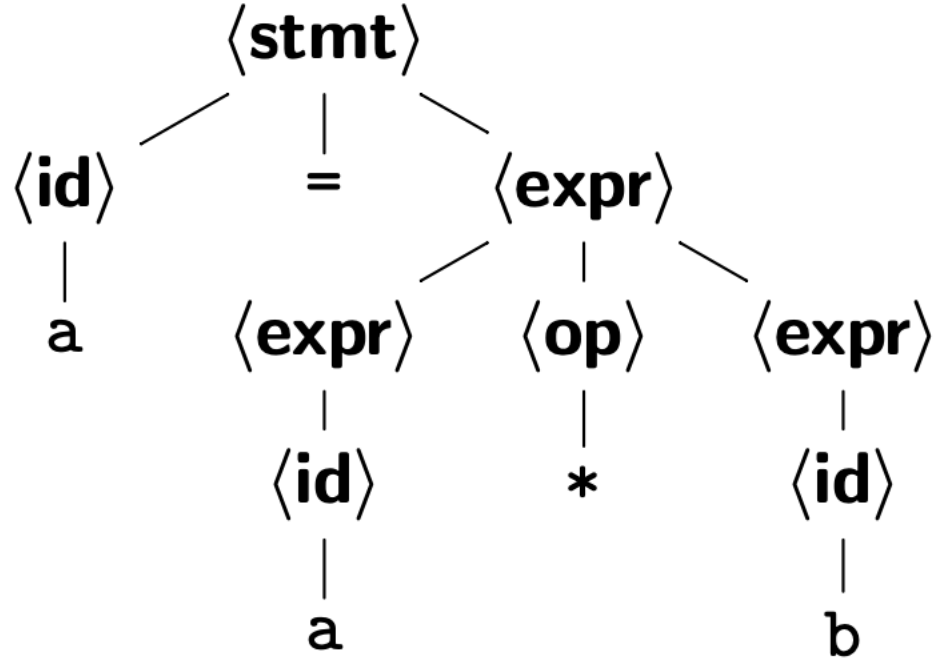
⟨stmt⟩ → ⟨id⟩ = ⟨expr⟩ → ⟨id⟩ = ⟨expr⟩ ⟨op⟩ ⟨expr⟩ → ⟨id⟩ = ⟨expr⟩ ⟨op⟩ ⟨id⟩

→ ⟨id⟩ = ⟨expr⟩ ⟨op⟩ b → ⟨id⟩ = ⟨expr⟩ * b → ⟨id⟩ = ⟨id⟩ * b → ⟨id⟩ = a * b

→ a = a * b

Çözümleme Ağacı (Parse Tree)

```
a = a * b
```



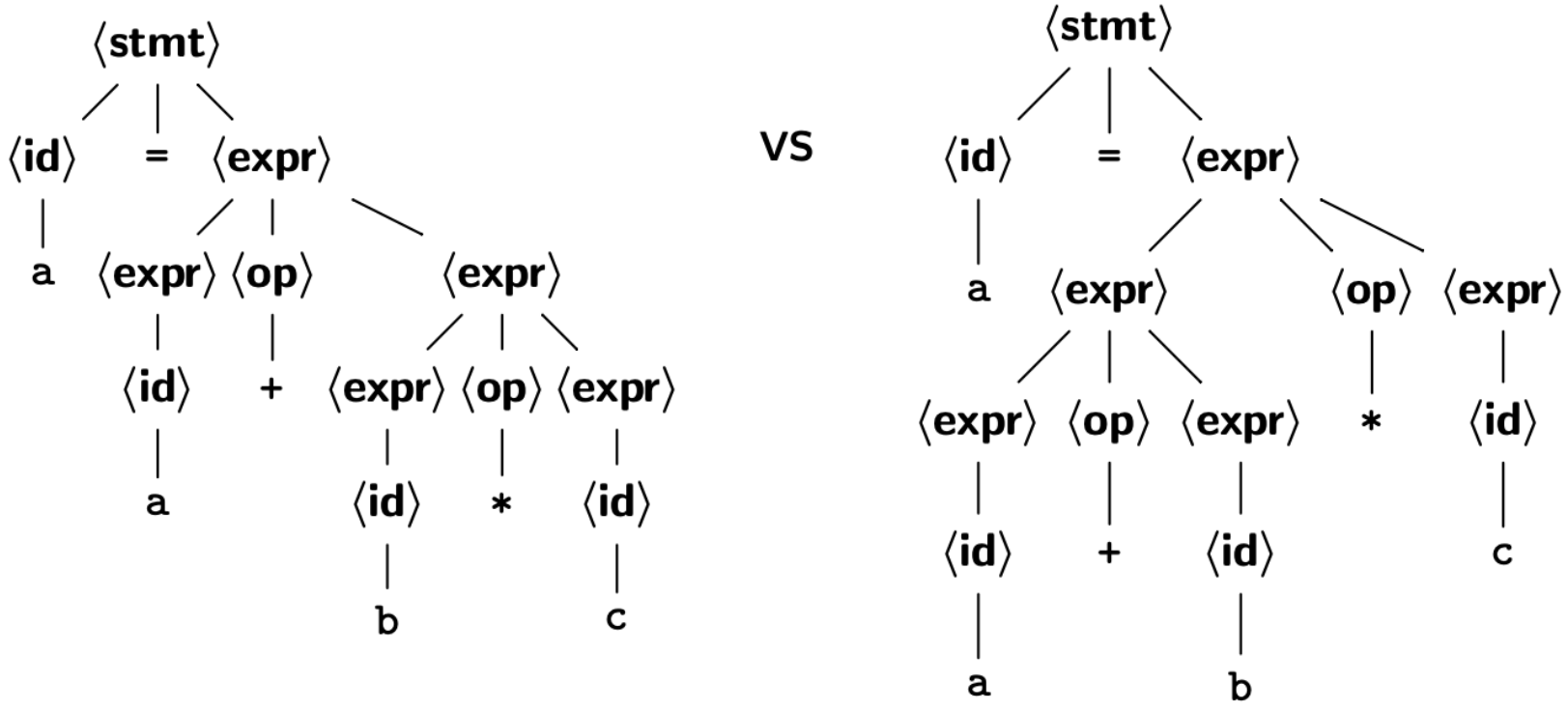
Ref: CENG 242 Ders Notları, Şehitoğlu, O. T.

Çözümleme Ağacı Üretimi

- Çözümleme ağacı, programın yapısını verir. Programın semantiği bu yapıyla ilişkilidir.
- Örneğin, yerel kapsamlar (local scopes), ifadelerin değerlendirme sırası vb.
- Derleme sırasında, kod oluşturma, semantik analiz ve optimizasyon işlemleri için çözümleme ağaçları gerekebilir.
- Bir çözümleme ağacı oluşturulduktan sonra, çeşitli derleme görevlerini yapmak için üzerinde gezinilebilir.

Muğlak (Ambiguous) Gramerler

a = a + b * c



Ref: CENG 242 Ders Notları, Şehitoğlu, O. T.

Muğlak (Ambiguous) Gramerler

- Aynı cümle farklı kurallar dizisi izlenerek türetilebilirse ve böylece farklı bir ayrıştırma ağacıyla sonuçlanırsa, bir gramer muğlak / belirsiz olarak adlandırılır.
- Çözümleme ağacı yapısı programın anlamsal yapısını değiştirirse, belirsizlik ciddi bir sorun haline gelir.
- Operatörlerin önceliği ifadenin değerini etkiler.
- Programlama dilleri, belirsizliği çözmek için öncelik kurallarını uygular.
- Çözüm:
  - Dilin gramer tasarımı belirsiz olmamalı.
  - Çözümleme sırasında, doğru çözümleme ağacını oluşturmak için kurallar belirlenmeli.

Öncelik (Precedence)

- Farklı öncelik seviyelerine sahip operatörler farklı şekilde ele alınmalıdır.
- Daha yüksek öncelikli işlemler çözümleme ağacının derinlerinde olmalıdır (yapraklara yakın).
- Daha düşük öncelikli işlemler köke daha yakın olmalıdır.
- Her öncelik düzeyi için bir sonlanmayan eleman (non terminal) tanımlanmalıdır.

Düzeltilmiş Gramer

a = a + b * c

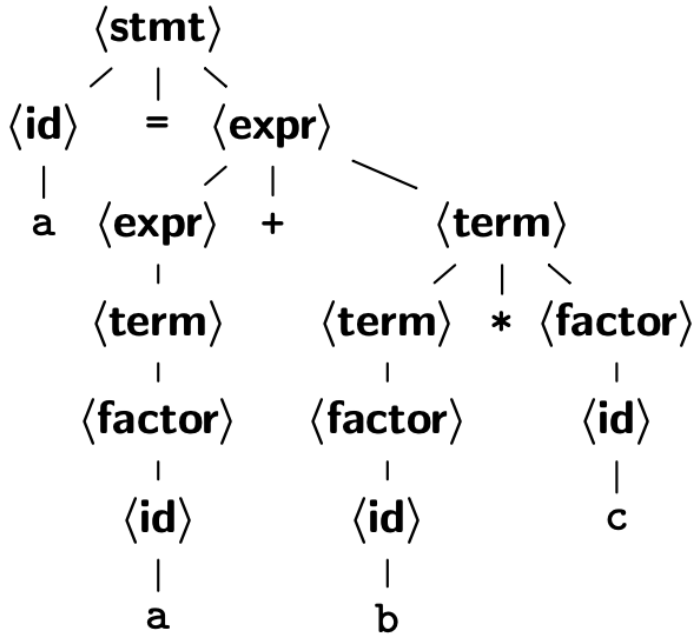
⟨stmt⟩ → ⟨id⟩ = ⟨expr⟩

⟨expr⟩ → ⟨expr⟩ + ⟨term⟩ | ⟨term⟩

⟨term⟩ → ⟨term⟩ * ⟨factor⟩ | ⟨factor⟩

⟨factor⟩ → ⟨id⟩ | (⟨expr⟩)

⟨id⟩ → a | b | c



- ⟨term⟩ ve ⟨expr⟩ farklı önceliklere sahip.
- ⟨term⟩ açılmaya başlayınca + asla ortaya çıkamaz.
- Sadece 1 şekilde çözümleme yapmak mümkündür.

Ref: CENG 242 Ders Notları, Şehitoğlu, O. T.