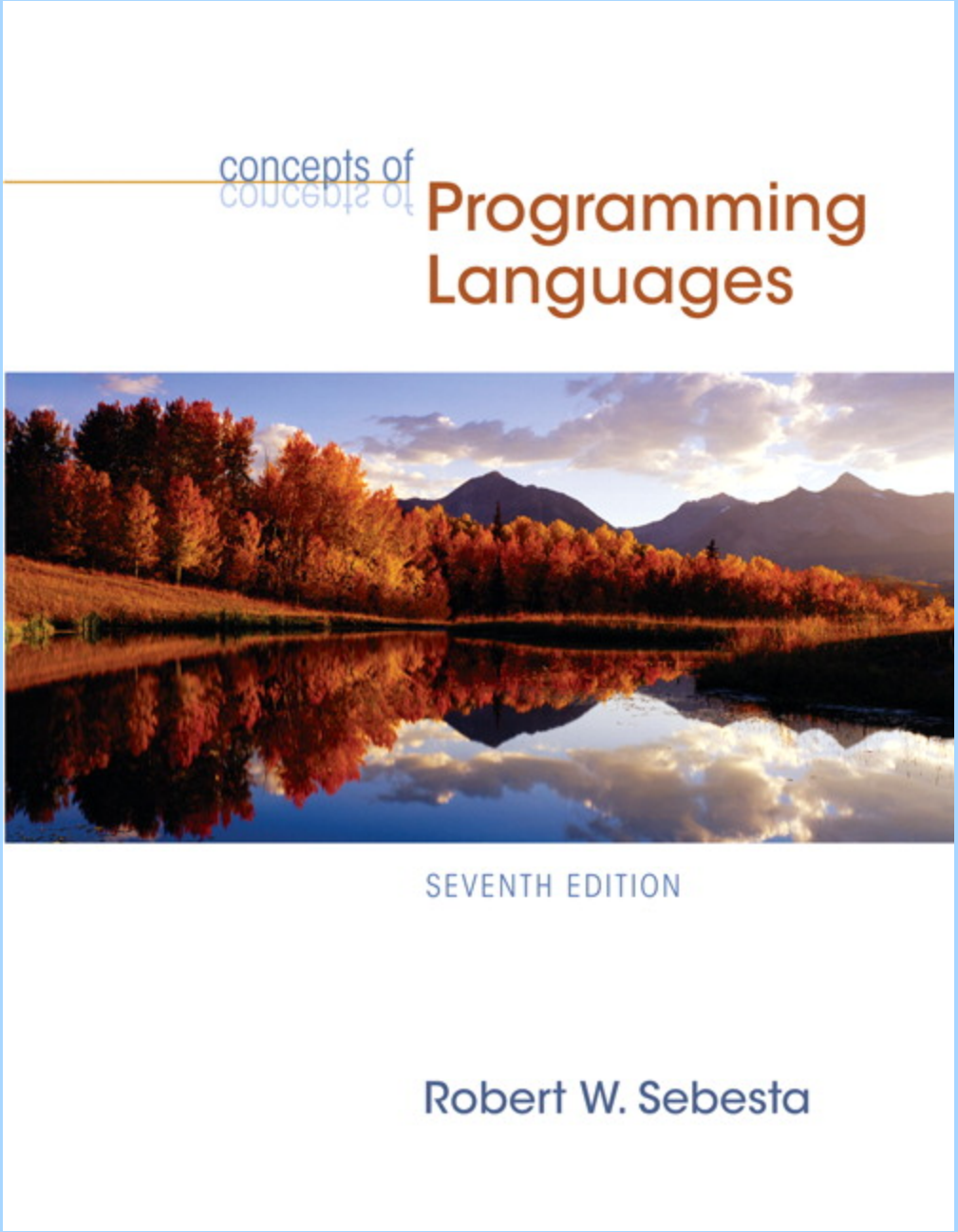
Bölüm 3

Sentaks ve Semantiği tanımlama



ISBN 0-321-33025-0

Bölüm 3 Konuları

1. Giriş
2. Genel Sentaks tanımlama problemi
3. Sentaks tanımlamanın biçimsel metotları
4. Özellik(Attribute) Gramerleri
5. Programların anlamlarını açıklamak: Dinamik Semantik

3.1 Giriş

- **Sentaks(Syntax)**: ifadelerin(statements), deyimlerin(expressions), ve program birimlerinin biçimi veya yapısı
- **Semantik(Semantics)**: deyimlerin, ifadelerin, ve program birimlerinin anlamı
- Sentaks ve semantik bir dilin tanımı sağlar
  - Bir dil tanımının kullanıcıları
    - Diğer dil tasarımcıları
    - Uygulamacılar(Implementers)
    - Programcılar (dilin kullanıcıları)

3.2 Genel Sentaks tanımlama problemi : Terminoloji

- Bir *cümle(sentence)* herhangi bir alfabede karakterlerden oluşan bir stringdir
- Bir *dil* (*language)* cümlelerden oluşan bir kümedir
- Bir *lexeme* bir dilin en alt seviyedeki sentaktik(syntactic) birimidir (örn., `*`, `sum`, `begin`)
- Bir *simge (token)* lexemelerin bir kategorisidir (örn., tanıtıcı(identifier))

Dillerin formal tanımları

- Tanıyıcılar(Recognizers)
  - Bir tanıma aygıtı bir dilin girdi strringlerini okur ve girdi stringinin dile ait olup olmadığına karar verir
  - Örnek: bir derleyicinin sentaks analizi kısmı
  - Bölüm 4'te daha detaylı anlatılacak
- Üreteçler(Generators)
  - Bir dilin cümlelerini(sentences) üreten aygıttır
  - Belli bir cümlenin(sentence) sentaksının doğru olup olmadığı, üretecin(generator) yapısıyla karşılaştırılarak anlaşılabilir

BNF ve Bağlam-Duyarsız (Context-Free) Gramerler

- Bağlam-Duyarsız(Context-Free) Gramerler
  - Noam Chomsky tarafından 1950lerin ortalarında geliştirildi
  - Dil üreteçleri(generators), doğal dillerin sentaksını tanımlama amacındaydı
  - Bağlam-Duyarsız(Context-Free) diller adı verilen bir diller sınıfı tanımlandı

Backus-Naur Form (BNF)

- Backus-Naur Form (1959)
  - John Backus tarafından Algol 58'i belirlemek için icat edildi
  - BNF bağlam-duyarsız (context-free) gramerlerin eşdeğeridir
  - BNF başka bir dili tanımlamak için kullanılan bir *metadil*dir(*metalanguage)*
  - BNF'de, soyutlamalar(abstractions) sentaktik(syntactic) yapı sınıflarını temsil etmek için kullanılır--sentaktik değişkenler gibi davranırlar (*nonterminal semboller* adı da verilir)

BNF'un Temelleri

- Non-terminaller(uçbirim olmayanlar): BNF soyutlamaları
- Terminaller(uçbirimler): lexemeler ve simgeler(tokens)
- Gramer: kurallar(rules) koleksiyonu
  - BNF kurallarından(rules) örnekler:

    <ident_list> → identifier | identifer, <ident_list>

    <if_stmt> → **if** <logic_expr> **then** <stmt>

BNF Kuralları(Rules)

- Bir kuralın bir sol-tarafı (left-hand side (LHS)) ve bir sağ tarafı (right-hand side (RHS)) vardır, ve *terminal* ve *nonterminal* sembollerden oluşur
- Bir gramer, kuralların(rules) boş olmayan sonlu bir kümesidir
- Bir soyutlama (abstraction)(veya nonterminal sembol) birden fazla RHS'ye sahip olabilir

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

Listeleri(Lists) Tanımlama

- Sentaktik listeler özyineleme(recursion) kullanılarak tanımlanır

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

- Bir türev(derivation), başlangıç sembolüyle başlayan ve bir cümleyle(sentence)(tüm terminal sembolleri)biten kuralların(rules) tekrarlamalı bir uygulamasıdır.

Bir Gramer Örneği

<program> → <stmts>

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <var> = <expr>

<var> → a | b | c | d

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

<term> → <var> | const

Bir türev(derivation) örneği

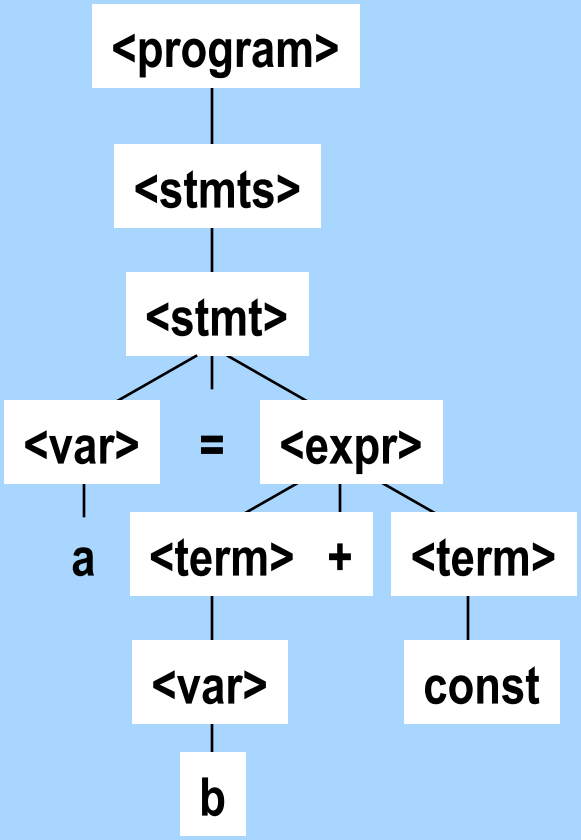
```
<program> => <stmts> => <stmt>
                     => <var> = <expr> => a =<expr>
                     => a = <term> + <term>
                     => a = <var> + <term>
                     => a = b + <term>
                     => a = b + const
```

Türev(Derivation)

- Bir türevde yer alan bütün sembol stringleri cümlesel biçimdedir(sentential form)
- Bir cümle(sentence) sadece terminal semboller içeren cümlesel bir biçimdir
- Bir ensol türev(leftmost derivation), içindeki her bir cümlesel biçimdeki ensol nonterminalin genişletilmiş olmadığı türevdir(is one in which the leftmost nonterminal in each sentential form is the one that is expanded)
- Bir türev(derivation) ensol(leftmost) veya ensağ (rightmost) dan her ikisi de olmayabilir

Ayrıştırma Ağacı (Parse Tree)

- Bir türevin(derivation) hiyerarşik gösterimi



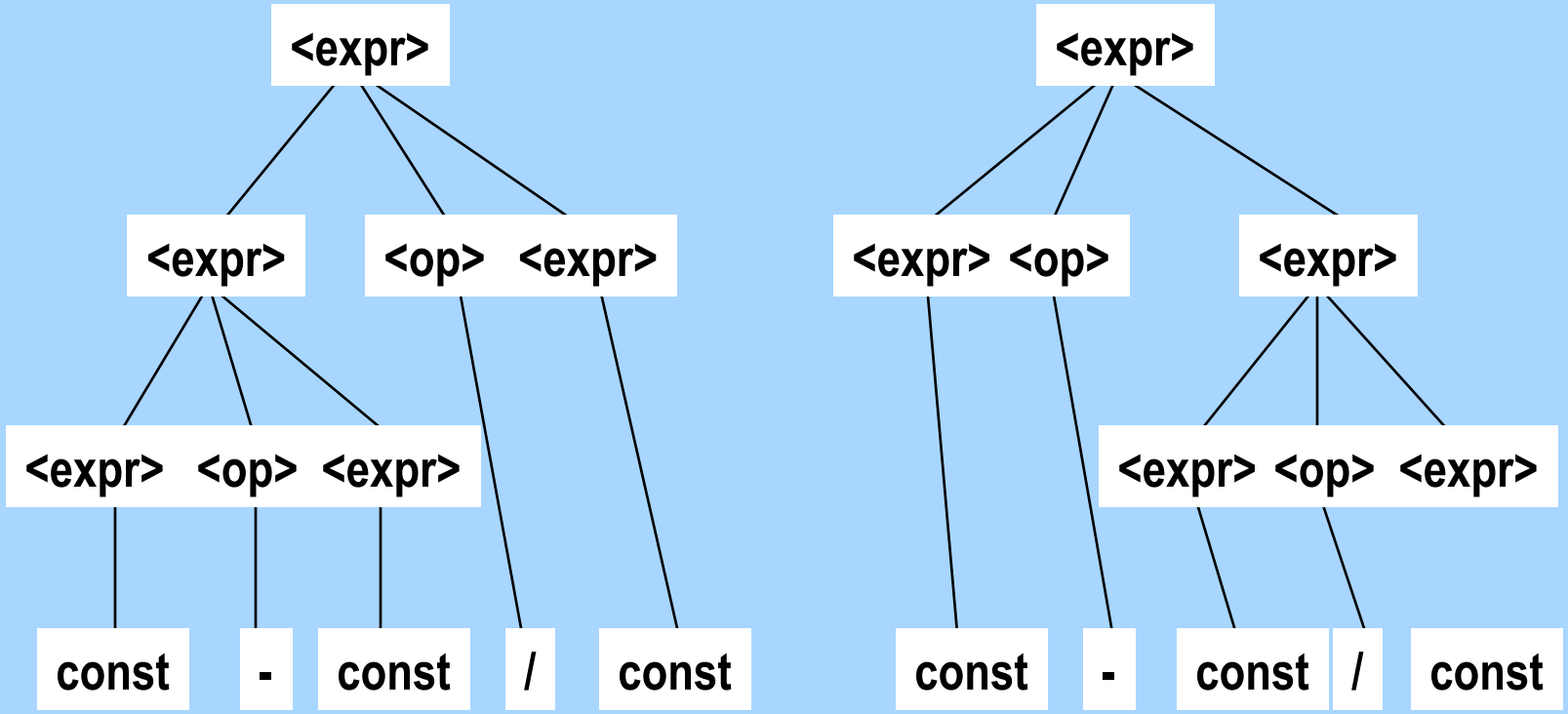
Gramerlerde Belirsizlik(Ambiguity)

- Bir grammer ancak ve ancak(iff) iki veya daha fazla farklı ayrıştırma ağacı(parse trees)olan bir cümlesel biçim(sentential form) üretiyorsa *belirsiz*dir(*ambiguous*)

Bir Belirsiz Deyim Grameri

<expr> → <expr> <op> <expr> | const

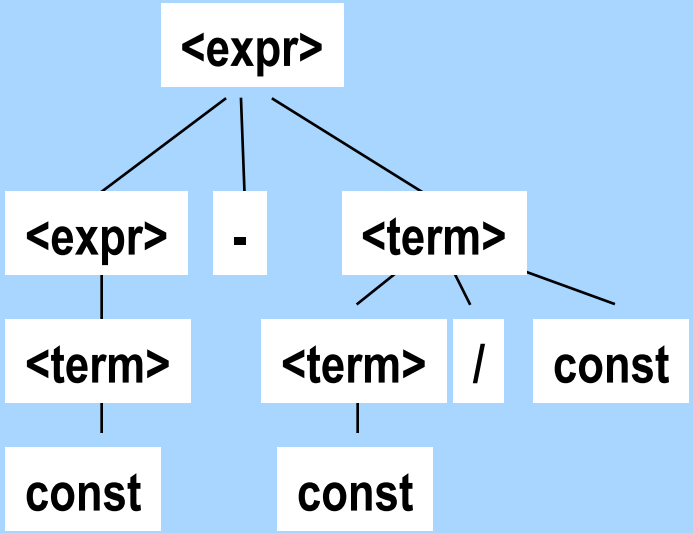
<op> → / | -



Bir belirsiz olmayan(Unambiguous) deyim Grameri

- Eğer ayrıştırma ağacını(parse tree) operatörlerin öncelik(precedence) seviyelerini göstermek için kullanırsak, belirsizlik elde edemeyiz

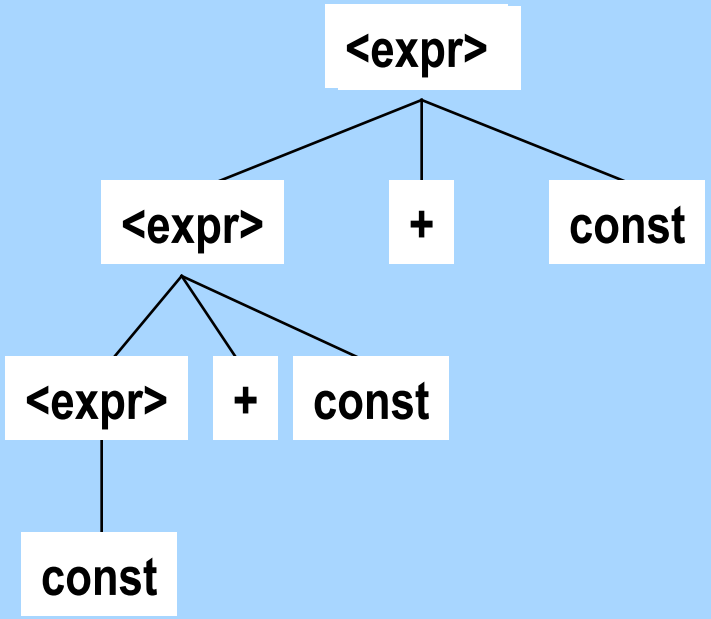
```
<expr> → <expr> - <term>  |  <term>
<term> → <term> / const| const
```



Operatörlerin Birleşirliği(Associativity)

- Operatör birleşirliği(associativity) de gramerle gösterilebilir

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



Extended BNF(Genişletilmiş BNF)

- Opsiyonel kısımlar köşeli parantez içine yerleştirilir ([ ])

  `<proc_call> -> ident [(<expr_list>)]`

- RHS lerin(sağ-taraf) alternatif kısımları parantezler içine yerleştirilir ve dikey çizgilerle ayrılır

  `<term> → <term> (+|-) const`

- Tekrarlamalar(Repetitions) (0 veya daha fazla) süslü parantez ({ }) içine yerleştirilir

  `<ident> → letter {letter|digit}` brace

BNF ve EBNF

- BNF

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

- EBNF

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

3.4 Özellik(Attribute) Gramerleri

- Bağlam-duyarsız(context-free) gramerler (CFGs) bir programlama dilinin bütün sentaksını tanımlayamazlar
- Ayrıştirma ağaçlarıyla(parse trees) birlikte bazı semantik bilgiyi taşıması için CFG'lere eklemeler
- Özellik(attribute) gramerlerinin(AGs) birincil değerleri :
  - Statik semantik belirtimi
  - Derleyici(Compiler) tasarımı (statik semantik kontrolü)

Özellik(Attribute) Gramerleri: Tanım

- Bir özellik grameri G = (S, N, T, P) aşağıdaki eklemelerle birlikte bir bağlam-duyarsız gramerdir :
  - Her bir *x* gramer sembolü için özellik değerlerinden(attribute values) oluşan bir *A(x)* kümesi vardır
  - Her kural(rule), içindeki nonterminallerin belirli özelliklerini(attributes) tanımlayan bir fonksiyonlar kümesine sahiptir
  - Her kural, özellik tutarlılığını(consistency) kontrol etmek için karşılaştırma belirtimlerinden(predicate) oluşan (boş olabilir) bir kümeye sahiptir

Özellik Gramerleri: Tanım

- $X_0 \rightarrow X_1 \ldots X_n$ bir kural olsun
- $S(X_0) = f(A(X_1), \ldots, A(X_n))$ biçimindeki fonksiyonlar *sentezlenmiş özellikleri* (*synthesized attributes*) tanımlar
- $I(X_j) = f(A(X_0), \ldots, A(X_n))$, $i <= j <= n$ için, şeklindeki fonksiyonlar *miras alınmış özellikleri* (*inherited attributes)* tanımlar
- Başlangıçta, yapraklarda *yerleşik özellikler(intrinsic attributes)* vardır

Özellik Gramerleri : Örnek

- Sentaks

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

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

  <var> A | B | C

- actual_type: <var> ve <expr> ile sentezlenmiştir
- expected_type: <expr> ile miras bırakılmıştır

3.4 Özellik Gramerleri (örn.)

- Sentaks kuralı: **<expr> → <var>[1] + <var>[2]**
  Semantik kurallar:

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

Karşılaştırma belirtimi (Predicate):

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

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

- Sentaks kuralı: **<var> → id**
  Semantik kuralı:

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

3.4 Özellik Gramerleri (örn.)

- Özellik(attribute) değerleri nasıl hesaplanır?
  - Eğer bütün özellikler miras alınmışsa, ağaç yukarıdan-aşağıya(top-down order) şekilde düzenlenir
  - Eğer özellikler sentezlenmişse, ağaç aşağıdan-yukarıya(bottom-up order) şekilde düzenlenir.
  - Çoğu kez, bu iki çeşit özelliğin her ikisi de kullanılır, ve aşağıdan-yukarıya ve yukarıdan-aşağıya düzenlerin kombinasyonu kullanılmalıdır.

3.4 Özellik Gramerleri (örn.)

<expr>.expected_type ← ebeveyninden miras almıştır

<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

3.5 Semantik

- Semantiği tanımlamak için yaygın kabul edilmiş tek bir gösterim veya formalizm yoktur
- İşlemsel(Operational) Semantik
  - Bir programı simulasyon veya gerçek olarak makine üzerinde çalıştırarak anlamını açıklamaktır. Makinenin durumundaki(state) değişme (bellek, saklayıcılar(registers), vs.) ifadenin anlamını tanımlar

3.5 Semantik (devamı)

- Yüksek-düzeyli bir dil için işlemsel semantiği kullanmak için, bir sanal (virtual) makine gereklidir
- Donanım saf yorumlayıcı(pure interpreter) çok pahalı olacaktır
- Yazılım saf yorumlayıcının bazı problemleri:
  - Bilgisayara özgü ayrıntılı özellikler faaliyetlerin anlaşılmasını zorlaştırır
  - Böyle bir semantik tanımı makine-bağımlı olurdu

İşlemsel(Operational) Semantik

- Daha iyi bir alternatif: Tam bir bilgisayar simulasyonu
- İşlem:
  - Bir çevirmen(translator) oluştur (kaynak kodunu, idealleştirilen bir bilgisayarın makine koduna çevirir)
  - İdealleştirilen bilgisayar için bir simülator oluştur
- İşlemsel semantiğin değerlendirmesi:
  - Informal olarak kullanılırsa iyidir(dil el kitapları, vs.)
  - Formal olarak kullanılırsa aşırı derecede karmaşıktır(örn., VDL), PL/I'ın semantiğini tanımlamak için kullanılıyordu.

3.5 Semantik

- Aksiyomatik(Axiomatic) Semantik
  - Biçimsel mantığa(formal logic) dayalıdır (predicate calculus)
  - Orjinal amaç: biçimsel program doğrulaması(verification)
  - Yaklaşım: Dildeki her bir ifade tipi için aksiyomlar veya çıkarsama kuralları(inference rules) tanımlamak(deyimlerin(expressions) diğer deyimlere dönüştürülmesine imkan sağlamak için)
  - Deyimlere iddia(assertions) adı verilir

Aksiyomatic Semantik

- Bir ifadenin önündeki bir iddia(assertion) (bir önşart(precondition)), çalıştırıldığı zaman değişkenler arasında true olan ilişki ve kısıtları(constraints) belirtir
- Bir ifadenin arkasından gelen iddiaya sonşart(postcondition) denir
- En zayıf önşart(weakest precondition), sonşartı garanti eden asgari kısıtlayıcı önşarttır

Aksiyomatik Semantik

- Ön-son(Pre-post) biçimi :

  {P} ifade {Q}

- Bir örnek: $a = b + 1$ $\{a > 1\}$

  Mümkün bir önşart: $\{b > 10\}$

  En zayıf önşart: $\{b > 0\}$

Aksiyomatik Semantik

- Program ispat(proof) işlemi: Bütün program için sonşart istenen sonuçtur. Programda ilk ifadeye kadar geriye doğru çalışılır. Birinci ifadedeki önşart program şartnamesiyle(spec.) aynıysa, program doğrudur.

Aksiyomatik Semantik

- Atama ifadeleri için bir aksiyomdur (x = E):

$$\{Q_{x->E}\}\ x = E\ \{Q\}$$

- Sonuç(Consequence) kuralı:

$$\frac{\{P\}\ S\ \{Q\},\ P' => P,\ Q => Q'}{\{P'\}\ S\ \{Q'\}}$$

Aksiyomatik Semantik

- Sıralar (sequences) için çıkarsama(inference) kuralı

  Bir S1;S2 sırası için:

  {P1} S1 {P2}

  {P2} S2 {P3}

Çıkarsama kuralı:

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

Aksiyomatik Semantik

- Mantıksal öntest döngüleri için bir çıkarsama kuralı

  Döngü yapısı için:

  {P} while B do S end {Q}

  çıkarsama kuralı:

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

  I döngü sabiti(invariant) ise. (tümevarımsal hipotez(inductive hypothesis))

Aksiyomatik Semantik

- Döngü sabitinin(invariant) özellikleri

  I , aşağıdaki şartları sağlamalıdır:

  - P => I (döngü değişkeni başlangıçta true olmalı)
  - {I} B {I} (Boolean hesabı I'nin doğruluğunu(geçerliliğini) değiştirmemelidir)
  - {I and B} S {I} (Döngü gövdesinin çalıştırılmasıyla I değişmez)
  - (I and (not B)) => Q (I true ise ve B false ise, Q bulunur(implied))
  - Döngü sonlanır (ispatlamak zor olabilir)

Aksiyomatik Semantik

- Döngü sabiti I, döngü sonşartının zayıflatılmış bir sürümüdür, ve aynı zamanda bir önşarttır.
- I, döngünün başlamasından önce yerine getirilebilecek kadar zayıf olmalıdır, fakat döngü çıkış şartıyla birleştirildiğinde, sonşartın doğru olmasını zorlayacak kadar güçlü olmalıdır

3.5 Semantik (devamı)

- Aksiyomatik Semantiğin değerlendirilmesi:
  - Bir dildeki bütün ifadeler için aksiyom ve çıkarsama(inference) kuralları geliştirmek zordur
  - İspatların doğruluğu için iyi bir araçtır, ve programlama mantığı için mükemmel bir çatıdır framework for reasoning about programs, fakat dil kullanıcıları ve derleyici yazanlar için kullanışlı değildir
  - Programlama dilinin anlamını tanımlamadaki yararlılığı dil kullanıcıları veya derleyici yazarları için sınırlandırılmıştır

3.5 Semantik (devamı)

- Denotasyonel Semantik
  - Özyinelemeli(recursive) fonksiyon teorisine dayalıdır
  - En soyut semantik tanımlama metodudur
  - Scott ve Strachey (1970) tarafından geliştirilmiştir

Denotasyonel(Denotational) Semantik

- Dil için denotasyonel şartnameler(spec) oluşturma işlemidir (kolay sayılmaz):
  - Her dil varlığı(entity) için matematiksel bir nesne tanımlanır
  - Dil varlıklarının(entity) örneklerini(instances) karşılık gelen matematiksel nesnelerin örneklerine eşleştiren bir fonksiyon tanımlanır
- Dil yapılarının(construct) anlamları sadece programın değişkenlerinin değerleriyle tanımlanır

3.5 Semantik (devamı)

- Denotasyonel and işlemsel(operational) semantik arasındaki fark: İşlemsel semantikte, durum(state) değişimleri kodlanmış algoritmalarla tanımlanır ; denotasyonel semantikte, sıkı matematiksel fonksiyonlarla tanımlanır

Denotasyonel Semantik

- Programın durumu(state) bütün güncel değişkenlerinin değerleridir

  $s = \{<i_1, v_1>, <i_2, v_2>, \ldots, <i_n, v_n>\}$

- **VARMAP** öyle bir fonksiyon olsun ki, bir değişkenin adı ve durumu verildiğinde, o değişkenin güncel değerini döndürsün

  $\mathbf{VARMAP}(i_j, s) = v_j$

3.5 Semantik (devamı)

- Ondalık sayılar
  - Şu denotasyonel semantik tanımı, string sembollerden oluşan ondalık sayıları sayısal değerlere eşleştirir

3.5 Semantik (devamı)

<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

3.5 Semantik (devamı)

- Deyimler(Expressions)
  - Deyimleri Z ∪ {error} üzerine eşleştirir
  - Deyimleri, ondalık sayılar, değişkenler, veya bir aritmetik operatör ve her biri bir deyim olabilen iki operanda sahip ikili(binary) deyimler olarak varsayıyoruz.

  We assume expressions are decimal numbers, variables, or binary expressions having one arithmetic operator and two operands, each of which can be an expression

3.5 Semantik (devamı)

```
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)
          ...
```

3.5 Semantik (devamı)

- Atama ifadeleri
  - Durum kümelerini durum kümelerine eşleştirir

$$
\begin{aligned}
&Ma(x := E, s)\ \Delta = \\
&\quad \text{if } Me(E, s) == error \\
&\qquad \text{then } error \\
&\qquad \text{else } s' = \\
&\{\langle i_1', v_1'\rangle, \langle i_2', v_2'\rangle, \ldots, \langle i_n', v_n'\rangle\}, \\
&\qquad \text{where for } j = 1, 2, \ldots, n, \\
&\qquad\quad v_j' = VARMAP(i_j, s) \text{ if } i_j <> x \\
&\qquad\qquad = Me(E, s) \text{ if } i_j == x
\end{aligned}
$$

3.5 Semantik (devamı)

- Mantıksal Öntest Döngüleri(Logical Pretest Loops)
  - Durum kümelerini durum kümelerine eşleştirir

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))

3.5 Semantik (devamı)

- Döngünün anlamı; program değişkenlerinin, döngüdeki ifadelerin belirtilen sayıda ve hata olmadığını varsayarak çalıştırılmasından sonra aldığı değerleridir
- Esasında döngü, iterasyondan özyinelemeye(recursion) dönüştürülmüştür, rekürsif kontrol(recursive control) matematiksel olarak diğer rekürsif durum eşleştirme fonksiyonlarıyla tanımlanır
- Özyineleme(Recursion), iterasyonla(iteration) karşılaştırıldığında, matematiksel kesinliklerle güçlüklerle (rigor) açıklaması daha kolaydır

3.5 Semantik (devamı)

- Denotasyonel semantiğin değerlendirilmesi
  - Programların doğruluğunu ispatlama için kullanılabilir
  - Programlar hakkında düşünmek için sıkı(kesin) (rigorous) bir yol sağlar
  - Dil tasarımında yardımcı olabilir
  - Derleyici üretme sistemlerinde kullanılmıştır
  - Karmaşıklığı yüzünden, dil kullanıcılaı tarafından çok az kullanılmıştır

Özet

- BNF ve bağlam-duyarsız gramerler eşdeğer meta-dillerdir
  - Programlama dillerinin sentaksını tanımlayabilmek için uygundur
- Özellik grameri bir dilin hem sentaksını hem de semantiğini tanımlayabilen tanımlayıcı bir formalizmdir
- Semantik tanımının birincil metotları
  - İşlem(Operation), aksiyomatik, denotasyonel

```
aa = "A";
bb = 3 * aa, "B";
cc = 3 * [aa], "C";
dd = {aa}, "D";
ee = aa, {aa}, "E";
ff = 3 * aa, 3 * [aa], "F";
gg = {3 * aa}, "D";
```

```
aa: A
bb: AAAB
cc: C AC AAC AAAC
dd: D AD AAD AAAD AAAAD etc.
ee: AE AAE AAAE AAAAE AAAAAE etc.
ff: AAAF AAAAF AAAAAF AAAAAAF
gg: D AAAD AAAAAAD etc.
```