BLM210 – HAFTA 3

ATTRIBUTE GRAMMARS, STATIC & DYNAMIC SEMANTICS

(ÖZELLİK GRAMERLERİ, STATİK (DURUK) VE DİNAMİK ANLAMBİLİM)

ATTRIBUTE GRAMMARS (ÖZELLİK GRAMERLERİ)

- **Attribute grammar** >> a device used to describe more of the structure of a programming language than can be described with a context-free grammar
  Özellik grameri >> bir programlama dilinin yapısına dair, bağlama duyarsız bir gramerin tanımlayabildiğinden daha fazlasını tanımlayabilen araç
  - An extension to a context-free grammar (Bağlama duyarsız gramere bir eklenti)
  - Describes rules such as type compatibility (tip uyumluluğu gibi kualları tanımlar)

STATIC SEMANTICS

- Rules that are difficult to describe with BNF
  BNF ile tarifi zor olan kurallar
  - The grammar may be too large to be useful (Gramer çok büyük ve kullanışsız olabilir)
    - It determines the size of the syntax analyzer
      Sözdizim çözümleyicinin büyüklüğü gramerin büyüklüğü ile orantılıdır.

- Rules that are impossible to describe with BNF
  BNF ile tarifi imkansız olan kurallar
  - Example: The rule that all variables must be declared before they are referenced
    Örnek: Bütün değişkenlerin kullanılmadan önce bildirilmesi gerekliliği kuralı

- Indirectly related to the meaning of programs during execution
  Programların koşma sırasındaki anlamlarına dolaylı ilişkisi var
- Related to legal forms of programs (syntax rather than semantics)
  Programların meşru biçimleriyle ilgili (anlamdan çok sözdizim ile)
- Rules such as type constraints (Tip kısıtları gibi kurallar)
- Named static because checking type specifications can be done at compile time
  Tip belirtim kontrolleri derleme zamanında yapıldığı için, statik (duruk) adı verilmiş

Attribute grammars >> formal approach both to describing and checking the correctness of the static semantic rules of a program
Özellik gramerleri >> Programların statik anlam kurallarını hem tanımlamak hem de kontrol etmek için kullanılan kurallı yaklaşım

Dynamic semantics >> meaning of expresssions, statements and program units
Dinamik anlam >> ifadeler, komutlar ve program birimlerinin anlamı

Basic concepts (Temel kavramlar)

Attribute grammars >> context-free grammars to which have been added attributes, attribute computation functions (semantic functions), and predicate functions
Özellik gramerleri >> bağlama duyarsız gramerlere; özellikler, özellik hesaplama fonksiyonları (anlamsal fonksiyonlar), ve yüklem fonksiyonlarının eklenmesiyle oluşan eklentili gramer

Attributes >> associated with grammar symbols (terminal and nonterminal) and have values assigned to them

Özellikler >> Gramer sembolleriyle (uç semboller ve dallanma sembolleri) ilişkili ve kendilerine atanan değere sahiptirler

Attribute computation functions (semantic functions) >> associated with grammar rules and specify how attribute values are computed

Özellik hesaplama fonksiyonları (anlamsal fonksiyonlar) >> gramer kurallarıyla ilişkilidirler ve özellik değerlerinin nasıl hesaplandığını belirtirler

Predicate functions >> associated with grammar rules and state the static semantic rules of the language

Yüklem fonksiyonları >> gramer kurallarıyla ilişkilidirler ve dilin static anlam kurallarını ifade ederler

<u>Attribute grammars defined (Özellik gramerlerinin tanımı)</u>

- Associated with each grammar symbol X (Her gramer sembolü X ile ilgili olarak)
  - A(X) >> a set of attributes (bir özellikler kümesi)
  - A(X) = S(X) ∪ I(X) where S(X) and I(X) are disjoint sets
    A(X) = S(X) ∪ I(X) >> S(X) ve I(X) ayrık kümeler
  - S(X) >> set of synthesized attributes (oluşturulmuş özellikler)
  - I(X) >> set of inherited attributes (edinilmiş özellikler)
  - *Synthesized attributes* >> to pass information up a parse tree
    Oluşturulmuş özellikler >> ayrışma ağacında bilgiyi yukarı geçirmek için
  - *Inherited attributes* >> to pass information down and across a parse tree
    Edinilmiş özellikler >> ayrışma ağacında bilgiyi aşağı ve yanlara geçirmek için

- Associated with each grammar rule (Her gramer kuralı ile ilgili olarak)
  - A set of semantic functions (bir anlamsal fonksiyonlar kümesi)

$$X_0 \rightarrow X_1 \ \ldots \ X_n$$
$$S(X_0) = f(A(X_1), \ \ldots \ , A(X_n))$$

$$X_j, 1 \leq j \leq n \qquad I(X_j) = f(A(X_0), \ \ldots \ , A(X_n))$$
$$I(X_j) = f(A(X_0), \ \ldots \ , A(X_{(j-1)}))$$

  - A possibly empty set of predicate functions
    Boş olması muhtemel bir yüklem fonksiyonları kümesi

    Below is the union of attribute sets (Aşağıdaki özellik kümelerinin birleşimidir)

$$\{A(X_0), \ \ldots \ , A(X_n)\}$$

    - Predicate function >> a Boolean expression on the union of the attribute set
      Yüklem fonksiyonu >> Özellik kümesinin birleşiminde bir Boolean ifadesi
    - Allowed derivations on an attribute grammar are those in which every predicate associated with every nonterminal is true.
      Bir özellik grameri üzerinde izin verilen türetmeler, her dallanma sembolü ile ilişkili her yüklemin doğru olduğu türetmelerdir.

- A false predicate function value indicates a violation of the syntax or static semantics rules of the language.
  Yanlış bir yüklem fonksiyonu değeri, dilin sözdizimsel veya statik anlam kurallarıyla ilgili bir ihlale işaret eder.

Parse tree of an attribute grammar >> parse tree based on its underlying BNF grammar, with a possibly empty set of attribute values attached to each node
Bir özellik gramerinin ayrışma ağacı >> Altta yatan BNF gramerini kullanan ve her sembole ait özellik değerlerini (muhtemelen boş olabilir) de gösteren ayrışma ağacı

- If all the attribute values in a parse tree have been computed, the tree is said to be **fully attributed**.
  Ayrışma ağacındaki bütün özellik değerleri hesaplanmışsa, ağaca **tam özelliklendirilmiş** denir.

Intrinsic attributes (Esas özellikler)

- Synthesized attributes of leaf nodes whose values are determined outside the parse tree
  Uç sembollere ait olup değerleri ayrışma ağacı dışında belirlenen oluşturulmuş özellikler

  - Ex (ör): the type of an instance of a variable (bir değişken örneğinin tipi)
    - possibly available in a look-up table (based on earlier declarations)
      bir tablodan çekilip elde edilebilir (önceki bildirimlerle belirlenmiş)

EXAMPLES OF ATTRIBUTE GRAMMARS (ÖZELLİK GRAMERİ ÖRNEKLERİ)

1) *The name on the* **end** *of an Ada procedure must match the procedure's name. (This rule cannot be stated in BNF.)*

Bir Ada altprogramının **end** *cümleciğindeki isim alprogramın ismiyle aynı olmalıdır. (Bu karal BNF'te ifade edilemez.)*

Syntax rule: <proc_def> → **procedure** <proc_name> [1]
<proc_body> **end** <proc_name> [2] ;
Predicate: <proc_name> [1] string == <proc_name> [2] .string

Syntax rule = sözdizim kuralı Predicate = yüklem Semantic rule = Anlamsal kural

2) *Checking type rules (tip kurallarının kontrolü)*

a. Simple assignment statements (Basit atama deyimleri)
b. Variable names are A, B, C. (Değişken adları A, B, C)
c. The right side of assignments can be a variable or an expression in the form of a variable added to another variable.
   Atamaların sağında bir değişken veya bir değişkenin başka bir değişkenle toplandığı bir ifade olabilir.
d. Variables can be either of type *int* or *real*. (Değişkenlerin tipi *int* veya *real* olabilir.)

e. When there are two variables on the right side of an assignments, they need not be of the same type.
Bir atamanın sağında iki değişken olduğunda, aynı tipte olmak zorunda değiller.
   i. If they are of different types, the type of the expression is *real*.
   Değişkenler farklı tipteyse, ifadenin tipi *real* olur.
   ii. If they are of the same type, the type of the expression is that same type.
   Değişkenler aynı tipteyse, ifadenin tipi bu aynı tip olur.

f. The type of the left side of the assignment must match the type of the right side.
Atamanın sol tarafının tipi, sağ tarafın tipiyle aynı olmalıdır.

Syntax portion of the attribute grammar (Özellik gramerinin sözdizim parçası)

<assign> → <var> = <expr>
<expr> → <var> + <var>
| <var>
<var> → A | B | C

Attributes for the nonterminals (Dalllanma simgelerinin özellikleri)

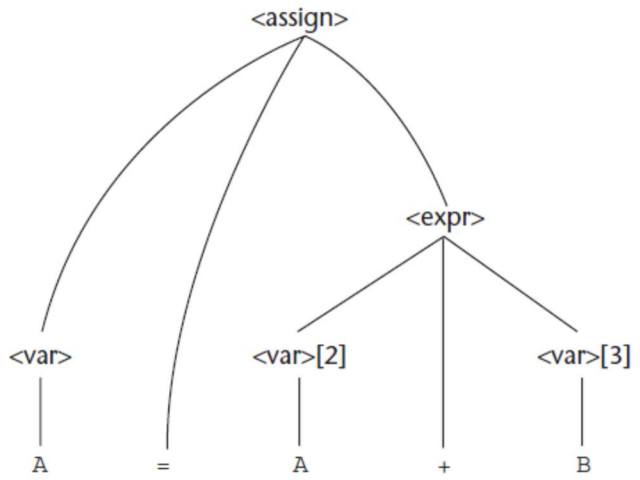
- *actual_type* >> synthesized attribute associated with the nonterminals <var> and <expr>
(gerçek tip) >> <var> and <expr> dallanma sembolleriyle ilişkili oluşturulmuş özellik
  - intrinsic for a variable (değişken için esas özellik)
  - for expressions, it is determined from actual types of child node (children nodes)
  ifadeler için, ifadenin çocuğu veya çocukları olan dallardan belirlenir.

- *expected_type* >> inherited attribute associated with the nonterminal <expr>
(beklenen tip) >> <expr> dallanma sembolüyle ilişkili edinilmiş özellik
  - determined by the LHS of an assignment (bir atamanın sol yanıyla belirlenir)

The following is the complete attribute grammar (Özellik gramerinin tümü aşağıdadır):

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → A | B | C
   Semantic rule: <var>.actual_type ← look-up (<var>.string)

A parse tree for** A = A+B **(**A = A+B **cümlesi için ayrışma ağacı)

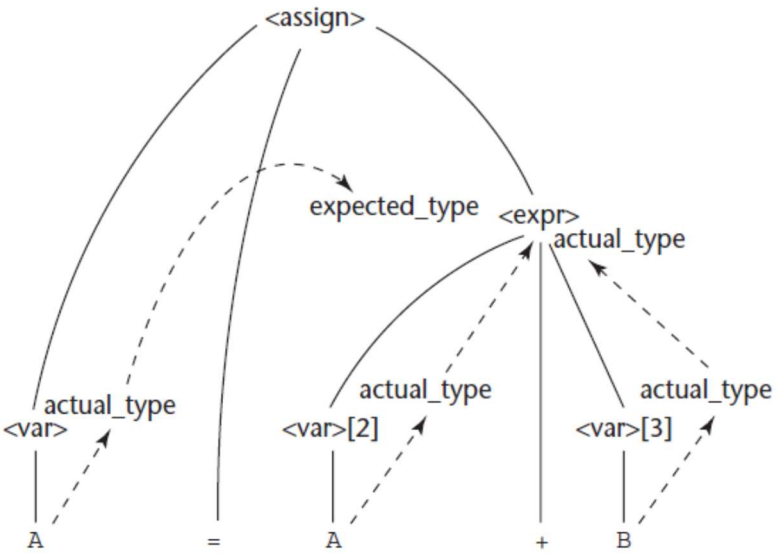


Computing attribute values (Özellik değerlerini hesaplama)

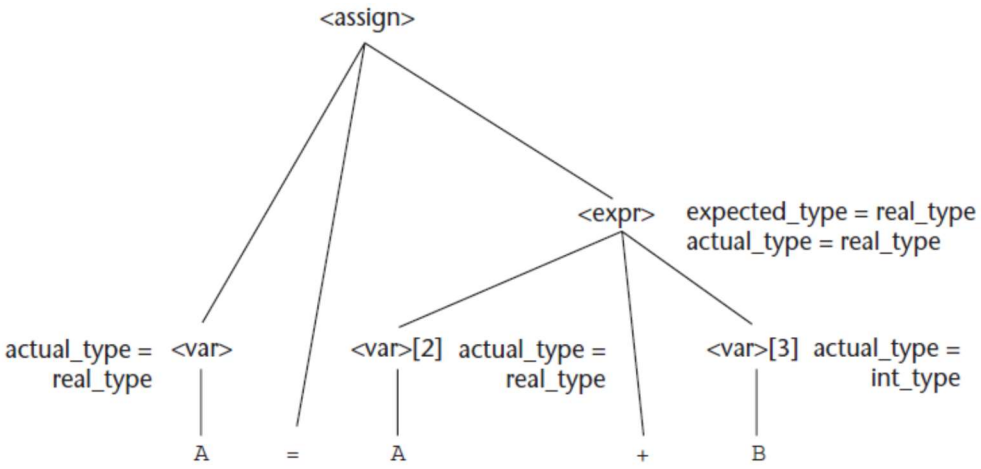
- Decorating the parse tree (ayrışma ağacını süslemek)
- top-down (if all attributes are inherited) or bottom-up (if all attributes are synthesized)
- yukarıdan-aşağı (bütün özellikler edinilmişse) veya aşağıdan-yukarı (bütün özellikler oluşturulmışsa)

1. <var>.actual_type ← look-up(A) (Rule 4)
2. <expr>.expected_type ← <var>.actual_type (Rule 1)
3. <var>[2].actual_type ← look-up(A) (Rule 4)
   <var>[3].actual_type ← look-up(B) (Rule 4)
4. <expr>.actual_type ← either int or real (Rule 2)
5. <expr>.expected_type == <expr>.actual_type is either TRUE or FALSE (Rule 2)

Flow of attributes in the tree (Özelliklerin ayrışma ağacındaki akışı)



Fully attributed parse tree (Tam özelliklendirilmiş ayrışma ağacı)



- Determining attribute evaluation order is a complex problem. It requires the construction of a dependency graph to show dependencies of all attribute dependencies.
  Özellik değeri hesaplamanın sırasını belirlemek, karmaşık bir problemdir. Bütün özellik bağımlılıklarını gösteren bir bağımlılık grafiğinin oluşturulması gerekir.

Evaluation of attribute grammars (Özellik gramerlerinin değerlendirilmesi)

- Checking the static semantic rules of a language is an essential part of all compilers.
  Dillerin statik anlam kurallarını kontrol etme, bütün derleyicilerin gerekli bir parçasıdır.

- Using an attribute grammar to describe all of the syntax and static semantics of a real contemporary programming language is the size and complexity of the grammar.
  Çağdaş bir dilin sözdizim ve statik anlam kurallarını tanımlamak için özellik gramerlerini kullanmak, gramerin büyüklüğü ve karmaşıklığı açısından sorundur.

- Less formal grammars are a powerful and commonly used tool for compiler writers, who are more interested in the process of producing a compiler than formalism.
  Biçimsellikten daha çok derleyici üretmeyi amaçlayan derleyici yazarları için, daha az biçimsel gramerler güçlü ve yaygın kullanılan araçlardır.

DYNAMIC SEMANTICS: Describing the meanings of programs
DİNAMİK ANLAMBİLİM: Programların anlamını tarif etmek

- Meaning of expressions, statements and program units
  İfadeler, deyimler ve program birimlerinin anlamı
- Describing syntax is relatively simple (Sözdizimi tanımlamak görece basittir)
- No universally accepted notation or approach has been devised for dynamic semantics
  Dinamik anlam için herkes tarafından kabul gören bir gösterim veya yaklaşım yoktur
- Software developers and compiler designers naturallly need to know what constructs in a programming language mean.
- Yazılım geliştiriciler ve derleyici tasarlayanlar, elbette programlama dillerindeki yapıların ne anlam ifade ettiğini bilmelidirler.

- If there were a precise semantic description of a programming language
  Bir programlama dilinin tam ve doğru bir anlamsal tanımı olsa

    - Programs could be proven correct without testing
      Programların doğruluğu deneme yapmadan kanıtlanabilirdi
    - Compilers could be shown to produce programs that exhibited exactly the behavior given in the language definition. (Their correctness could be verified)
      Derleyicilerin dil tanımında sözü edildiği şekilde davranan programlar ürettiği gösterilebilirdi. (Doğrulukları onaylanabilirdi)
    - Semantic description could be used as a tool to generate compilers automatically.
      Derleyicileri otomatik olarak üretmek için dinamik anlam tanımı kullanılabilirdi.
    - Language designers, who would develop the semantic descriptions of their languages, could discover ambiguities and inconsistencies in their designs.
    - Tasarladıkları dilin anlamsal tanımını oluşturacak olan dil tasarımcıları, tasarımlarındaki belirsizlik ve tutarsızlıkları belirleyebilirdi.

Operational semantics (İşlemsel anlam)

>> describe the meaning of a statement or program by specifying the effects of running it on a machine
>> makinede koştuğunda oluşturduğu etkileri belirterek, deyim ya da programların anlamını tarif etmek

>> a sequence of changes in the state of a machine (makinenin durumunda olan değişiklikler dizisi)

state >> a collection of values in the machine's storage
durum >> makinenin belleğindeki değerlerin tümü

- An obvious operational semantics description is given by executing a compiled version of the program on a computer.
- Açık bir işlemsel anlam tanımı, programın derlenen halini bilgisayarda koşarak elde edilir.

- <u>Problems (sorunlar)</u>

    - The individual steps in the execution of machine language and the resulting changes to the state of the machine are too small and too numerous.
      Makine dilindeki programın koşması ve sonuç olarak makinenin durumunda oluşan değişiklikler çok küçük ve sayıca fazladır.
    - The storage of a real computer is too large and complex. There are usually several levels of memory devices, as well as connections to other computers and memory devices through networks.
      Gerçek bir bilgisayarın belleği çok büyük ve karmaşıktır. Genelde, birden fazla düzeyde bellek aracıyla beraber ağlar üzerinden diğer bilgisayarlar ve bellek araçlarına bağlantılar vardır.
    - Machine languages and real computers are not used in formal operational semantics. Rather, intermediate-level languages and interpreters for idealized computers are designed for this purpose.
      Makine dili ve gerçek bilgisayarlar kurallı işlemsel anlam için kullanılmaz. Bunun yerine, ara-düzey diller ve ideal bilgisayar gibi davranan anlamlandırıcılar, amaç doğrultusunda tasarlanmıştır.

- Levels of operational semantics (işlemsel anlamın düzeyleri)

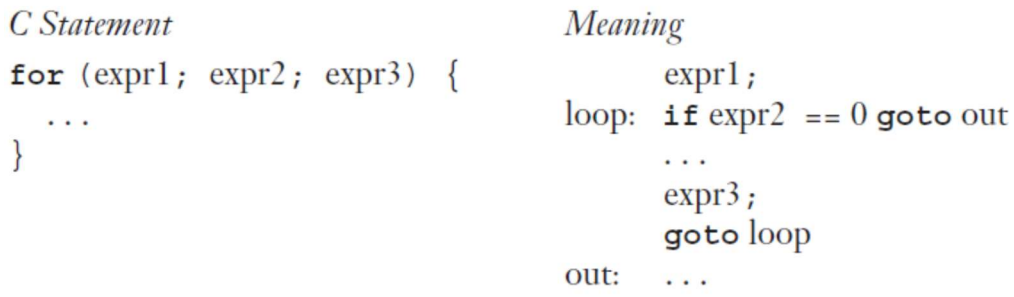
  - Natural operational semantics (doğal işlemsel anlam)
    - final result of the execution of a program
    - program koşulunca ortaya çıkan sonuç

  - Structural operational semantics (yapısal işlemsel anlam)
    - precise meaning of a program through an examination of the complete sequence of state changes that occur when the program is executed
      program koşulduğunda oluşan bütün durum değişikliklerini inceleyerek programın tam anlamına hakim olmak

- Basic process (temel süreç)

  - Construct an intermediate language (machine language is too low-level)
    Ara bir dil oluştur (makine dili çok alt-düzey)
  - Construct a virtual machine (interpreter) for the intermediate language
    Ara dil için sanal bir makine (anlamlandırıcı) oluştur
  - The virtual machine can be used to execute either single statements, code segments, or whole programs.
    Sanal makine tekli deyimleri, kod parçalarını veya tüm programı koşmak için kullanılabilir.
  - The semantics description can be used without a virtual machine if the meaning of a single statement is all that is required. In this use, which is structural operational semantics, the intermediate code can be visually inspected.
    Gereken tek bir deyimin anlamıysa, anlamsal tanım sanal makine olmadan kullanılabilir. Bu kullanımda (yapısal işlem anlamını tanımlamak için), ara kod gözle incelenebilir.

  - Example (Örnek)



C Statement	*Meaning*
`for (expr1; expr2; expr3) {`	`expr1;`
`...`	`loop: if expr2 == 0 goto out`
`}`	`...`
	`expr3;`
	`goto loop`
	`out: ...`

    - The virtual computer is the human reader. He/she can execute the instructions and observe the results.
      Sanal bilgisayar, insan olan okuyucudur. Komutları gözüyle koşup sonuçları gözlemleyebilir.

  - Intermediate languages and their associated virtual machines are highly abstract.
  - Ara diller ve onların sanal makineleri, yüksek düzeyde soyuttur.
  - The intermediate language should be convenient for virtual machines rather than humans.
  - Ara dil insanlardan daha çok, sanal makineler için elverişli olmalı.

- Example for humans (İnsanlar için örnek)

  - Describing simple control statements (Basit kontrol deyimlerini tanımlama)

    ident = var
    ident = ident + 1
    ident = ident – 1
    goto label
    if var relop var **goto** label

    relop >> { =, <>,>, <, >=, <=}

  - A slight generalization (Küçük bir genelleştirme)

    ident = var bin_op var
    ident = un_op var

- Evaluation (değerlendirme)

  - VDL = Vienna Description Language
  - Example (örnek): The VDL description of PL/I (PL/I'ın VDL tanımı)

  - Operational semantics provides an effective means of describing semantics for language users and language implementors, as long as the descriptions are kept simple and informal. Tanımlamalar basit ve kuralsız olduğu müddetçe; işlemsel anlam, dili kullananlar ve gerçekleyenler için anlamı tanımlamak için etkin bir araç sunar.

  - Operational semantics depends on languages of lower levels, not mathematics, which can sometimes lead to circularities in descriptions.
    İşlemsel anlam matematiğe değil, daha alt-düzeydeki dillere bağlıdır. Bu, bazen tanımlamalarda çemberselliğe (kendini kendiyle tanımlama) yol açabilir.

Denotational semantics (Gösterimsel anlam)

>> the most rigorous and most widely known formal method to describe the meaning of programs
>> programların anlamını tanımlamada en katı ve en çok bilinen yöntem

>> based on recursive function theory (özyinelemeli fonksiyon kuramı üzerine kurulu)

>> define for each language entity both a mathematical object and a function that maps instances of that language entity onto instances of the mathematical object.
>> her dil öğesi için hem bir matematiksel nesne hem de o dil öğesini matematiksel nesnenin örnekleriyle eşleyen bir fonksiyon tanımlanır.

>> Mathematical objects model (denote) the exact meaning of their corresponding entities.
>> Matematik nesneleri, karşılık gelen öğelerin tam anlamını modellerler (gösterirler).

>> Domain and range of mapping functions (Eşleme fonksiyonlarının tanım ve değer kümeleri var.)

  - Domain >> syntactic domain (syntactic structures that are mapped)
    Tanım kümesi >> sözdizimsel tanım kümesi (eşlenen sözdizimsel yapılar)

- Range >> semantic domain (mathematical objects to which syntactic structures are mapped)
  Değer kümesi >> anlamsal değer kümesi (sözdizimsel yapıların eşlendiği matematiksel nesneler)

>> Denotational semantics is related to operational semantics.
>> Gösterimsel anlam, işlemsel anlamla ilişkilidir.

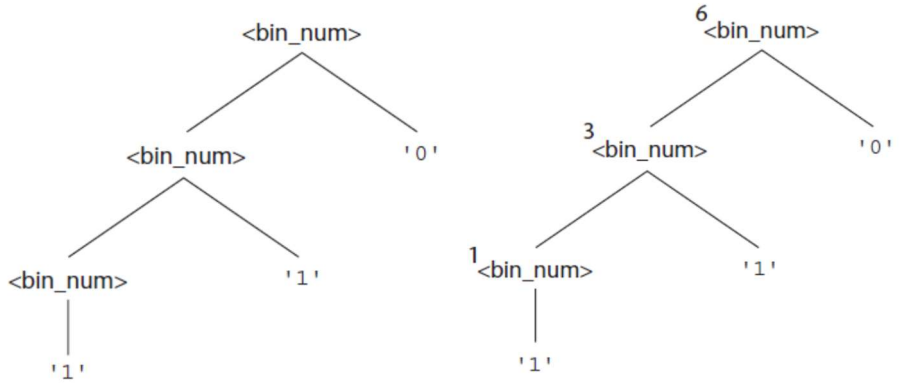
- In operational semantics, programming language constructs are translated into simpler programming language constructs, which become the basis of the meaning of the construct.
  İşlemsel anlamda, programlama dili yapıtaşları, yapının anlamına temel oluşturacak daha basit programlama dili yapıtaşlarına çevrilirler.

- In denotational semantics, programming language constructs are mapped to mathematical objects, either sets or, more often, functions.
  Gösterimsel anlamda, programlama dili yapıtaşları matematiksel nesnelere (kümeler ya da çoğunlukla fonksiyonlar) eşlenirler.

- No step-by-step computational processing of programs in denotational semantics
  Gösterimsel anlam, programların adım-adım koşma süreciyle ilgilenmez.

- Example (örnek)

<bin_num> → '0'
| '1'
| <bin_num> '0'
| <bin_num> '1'



M_{bin}('0') = 0
M_{bin}('1') = 1
M_{bin}(<bin_num> '0') = 2 * M_{bin}(<bin_num>)
M_{bin}(<bin_num> '1') = 2 * M_{bin}(<bin_num>) + 1

M_{bin} is the semantic function that maps a binary number to its decimal equivalent.
M_{bin}, ikili tabandaki bir sayıyı, karşılık gelen onluk bir sayıya eşleyen anlamsal fonksiyondur.

- Example (örnek)

<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}('2') = 2, . . ., 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

- The state of a program (Bir programın durumu)
  - s >> state of a program (durum)
  - i >> variables (değişkenler) v >> associated values (karşılık gelen değerler)

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

  - VARMAP >> a function of a variable name i_j and a state s
    >> bir değişken adı i_j ve bir durum s'nin bir fonksiyonu
    >> returns the associated value v_j of i_j
    >> değişken i_j'ye karşılık gelen değer v_j'yi dönüyor

    VARMAP(i_j, s) = = > v_j

  - Most semantic mapping functions for programs, program constructs map states to states.
    Program ve program yapıtaşlarına ait çoğu eşleme fonksiyonu durumu duruma eşler.

  - These state changes are used to define the meanings of programs, program constructs.
    Bu durum değişiklikleri program ve program yapılarının anlamını tanımlar.

  - Some language constructs—for example, expressions—are mapped to values, not states.
    Bazı dil yapıtaşları –örneğin, ifadeler- durumlara değil, değerlere eşlenir.

- Expressions (ifadeler)

<expr> → <dec_num> | <var> | <binary_expr>
<binary_expr> → <left_expr> <operator> <right_expr>
<left_expr> → <dec_num> | <var>
<right_expr> → <dec_num> | <var>
<operator> → + | *

  - **Z ∪ error** >> semantic domain (anlamsal tanım kümesi)

M_e (<expr>, s) $\Delta=$ case <expr> of
<dec_num>=>M_{dec} (<dec_num>, s)
<var> =>if VARMAP (<var>, s) == **undef**
then **error**
else VARMAP (<var>, s)
<binary_expr> =>
if(M_e (<binary_expr>.<left_expr>,s) == **undef** OR
M_e (<binary_expr>.<right_expr>, s) == **undef**)
then **error**
else if (<binary_expr>.<operator> == '+')
then M_e (<binary_expr>.<left_expr>, s) +
M_e (<binary_expr>.<right_expr>, s)
else M_e (<binary_expr>.<left_expr>, s) *
M_e (<binary_expr>.<right_expr>, s)

- Assignment statements (atama deyimleri)

M_a (x = E, s) $\Delta=$ 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)

  - i_j == x >> This comparison is of names, not values.
    >> Bu karşılaştırma değerleri değil, isimleri karşılaştırıyor.

- Logical pretest loops (mantıksal ön sınamalı döngü)

  - M_{sl} >> maps statement lists and states to states or **error**
    >> deyim listeleri ve durumları, durumlara veya **hata**ya eşler.
  - M_b >> maps Boolean values to Boolean values or **error**
    >> Boolean değerleri Boolean değerlere veya **hata**ya eşler.

M_l (**while** B **do** L, s) $\Delta=$ 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))

  - Note use of recursion (özyinelemeye dikkat edelim)
  - Loop may compute nothing because of nontermination
    Döngü sonlanmama dolayısıyla hiçbir şey hesaplamayabilir.

- Evaluation (değerlendirme)

    - Denotational semantics provides a framework for thinking about programming in a rigorous way.
      Gösterimsel anlam, programlamayı katı biçimde düşünmek için bir çerçeve oluşturuyor.

    - Statements for which denotational semantics is complex and difficult for language implementors may indicate that such statements are also difficult for language users and that alternative design is in order.
      Dil gerçekleyiciler açısından, deyimlere ait gösterimsel anlamın karmaşık ve zor olması, bu deyimlerin dili kullananlar açısından da zorluğuna işaret eder. Bu durumda, başka bir tasarım uygun olabilir.

    - Because of the complexity of denotational descriptions, they are of little use to language users. On the other hand, they provide an excellent way to describe a language concisely.
      Gösterimsel anlamın karmaşıklığından dolayı, kullanışlılığı azdır. Öte yandan, bir dili özlü bir biçimde tanımlamak için mükemmel araçlardır.