Type checking

→ subprogram and assignment'ta type cheking yapilir.

→ coercion: automatic conversion

Strongly typing

→ hatalarin tamamini yakalar.

avantaj: reliability

dezavantaj: - kodu uzatir.
- gelişmeyi yavaşlatir.
- yazmayi yavaşlatir.

→ static binding = static type cheking

→ dynamic binding = dynamic type cheking

unions are not type checked...

*Sair diyorki;
if the grammar ise unambiguous, then every right sentential form of the grammar has exactly one handle

● Operator Associativity
↳ left-associative: addition, (-), (*) and (/)
↳ right-associative: exp, assignment and conditional

● Attribute grammar:
defines the meaning for the production rules which gives the meaning for the input program.

Scope:

→ lokal variable nerede tanimlanmiş ise orada görülebilirdir.

→ shadow variable ? shallow

→ Scope evaluation: dynamic scope'un avantajlari: convenience ve flexibility
dezavantaji ise: poor readability (same name, different non-locals)
reliability
longer access times for non-local variables

→ Referencing Environment:
All names that are visible in the statement

*Type constructor → Cartesian product, - Union, - Subset
→ Function (Arrays)

— Type —
Value üzerindeki operasyon kümesi

Union: data items with different types are stored in overlapping region, reduce memory allocation
→ only one type of value is valid at one time

7.2 bol

Discriminated unions:
→ diğer isimleri disjoint, variant type or algebric type union
→ data abstraction için elverişlidir.

Vector - List

Vectors: like arrays, more flexibility, especially dynamic resizability

Lists: like vectors, can only be accessed by counting down from the first element

Polymorphism vs Overloading

● parametric polymorphism
→ single algorithm may be given many types
→ type variable may be replaced by any type
f: t→t => f: int→int

● Overloading
A single symbol may refer to more than one algorithm
Each algorithm may have different type

* C and C++
Arraylar için yer stackten ayrilir.

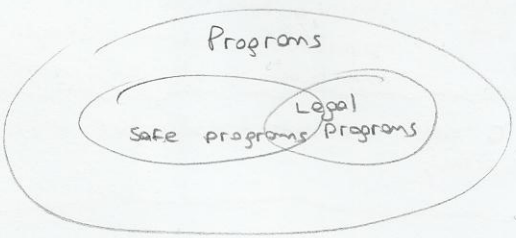
Pointer → Advantages
→ Adressing flexibility
→ recursive data structure
→ dynamic storage management

Overloading

Overloading vs polymorphism?

— Type inference?
— Type Equivalance?
— Type Compatibility?

● C, Pascal, C++: all type checked at compilation time



Syntax

1) Lexical syntax
: Basic sembol (name, value, ...)

2) Concrete syntax
- Rules for writing statevent, programs

3) Abstract syntax
→ meaning (semantic)

Chomsky hierarchy

1) Regular grammar
: used to define lexial structure of the language

A → w or A → Bw
(left regular)

A → w or A → wB
(right regular)

2) Context-free grammar
: used to define concrete syntax of the language

3) Context-sensitive grammar
: Unrestricted grammars
: recursively enumerable languages

⇒ Used to describe concrete syntax
→ Typically using BNF notation

Context Free Grammer

→ is a set of recursive productions rules used to generate pattern of string

Parse Tree

: 2 şekilde oluşturulur.

1) Top-down
: recursive descent parsing

2) Bottom up
: shift reduce parsing

Shift reduce parsing

There are four possible actions

- Shift , - Reduce , - Accept , - Error

→ initial stack empty
→ end of input is empty

advantage of implicit declaration

→ writability

disadvantage:
→ reliability

→ Context-Free Grammars

→ Used to describe concrete syntax

Difference

- OS ; coded algorithms causes state changes
- DS : state chages are defined by mathematical changes
- OS : state of the machine
- DS : state of the program

Traversing a DFA

→ Configuration : stak + remaining input

● Static variables : compile anında memory cell'lerine bound edilirler.
- program sonlanıncaya kadar orada dururlar

avantaj	dezavantaj
efficiency	less flexible
history sensitive (→ C static variable)	

● Stack dynamic variables :
→ Allocated from the run-time stack
→ De-allocate after execution
→ can happen at the beginning of block or anywhere (C method variable)

advantage	disadvantage
→ allows recursion	→ subprograms cannot be history sensitive
→ conserves storages allocate when and where needed	→ indirect addressing (longer time)
	→ overhead of allocation, deallocation

● Explicit Heap-Dynamic Variables :
→ Allocated and de-allocated by explicit directives
→ takes effect during execution
→ sadece reference ve pointer ile yapılır.

advantage :	disadvantage
flexibility (new and delete)	- reliability must be shown - inefficient by comparison

→ java ve C 'deki variable tanımlamaları statik olarak bound ediliyormuş ?

● Implicit heap dynamic variables
→ value assign edildiğinde otomatik olarak heape atılır. (bound edilir)
→ Allocation and de-allocation caused by assignment

advantage	disadvantage
- Flexibility - writability	- inefficient, because dynamic → error detection zor.

(array in Javascript)

Type checking

→ Operatör ile operandın uygunluğuna bakar.
→ Subprogramlar ve assignmentte type checking yapılır.
→ Coercion : implicit type checking

Type Compability 2 çeşit compability vardır.

- Name comp
- Structure type comp.

1) Name type comp : same declaration. yada aynı declarationı kullanan tip isimleri olmalı, impl. kolaydır.

2) Structure type comp: Aynı struct yapısına sahip ise structural comp. var deriz. flexible fakat impl. zordur.

Dynamic scope yazılış sekliyle alakalı değil çağrılma sırasına göre. çağırıldığı yerden geriye giderek arar.

→ scope : statik ve localdir. lifetime program boyunadır.

Uses for Types

Enumerated types

→ Ordered set whose elements are named and listed explicitly

avantaj: efficiency, readability, maintainability reliablity

Strong typing:

→ Strongly type ise yapılan hataları detect eder. Bütün hataları yakalarsa biz buna strongly typing diyoruz.

avantaj: reliability
dezavantaj: kodu uzatır, daha yavaş genişletir yazılabilirliği azaltır.

Scope:

→ değişkenlerin görüldüğü statementlerin aralığıdır.

Static scope: önce lokale bak, sonra scopu genişlet. ilk bulduğun declaration

Referencing environments: Değişkenin görülebilir olduğu cümledir.

→ **Discriminated Unions** Farklı tipleri bir arada tutabilir, Data abstraction yapar. Union type'da aynı anda tek değer bound edilir. Polymorphism destekler.

Polymorphic: Operatör ve fonksiyon farklı tiplerden herhangi biri ile çalışabilir. Re-use sağlanır. Genericler farklı tipler üzerinde çalışırlar.

Type checking: **C's name equivalence**

1) Type Inference
2) Type Equivalence : 2 tip aynı mı diye bakar.
3) Type Compatibility : Birbirinin yerine kullanımı

Type inference: kodda bakmadan oluşan çıktıya yada return valuesine göre checking yapılır.
→ type declarationuna gerek yoktur.
→ hata yakalanabilir.

→ Implicit conversion : upcasting
→ Explicit " : downcasting

Expression	Statements
→ No side effect	→ side effect
→ Return value	→ No return value

→ BNF kullanılarak syntax tanımlanır. Generation: dilin üretimi, Recognition: dilin tanınması

→ Elimizdeki yapıları kullanarak o dilde ifade edebileceğimiz tüm yapıları üretebiliyoruz. (Generation)

→ Hepsinin hangi dilde yazıldığını ve hiçbir dile uyup uymadığını bulmak recognition oluyor.

~ Chapter 4 (Names, Binding, Scope) ~

Dynamic semantic: programlama dilinin expression, statement ve program unitleri olarak tanımlanır.

Operational semantic: Run time sırasında machine'de olan etkilerin belirtileri.
- state change'lere yer verir.

Denotational semantik: anlamın tanımlanması için yapılan formal yöntemler.
→ recursive olaylar.
→ matematiksel model.

⇒ Differences:
O.S: state change sebep olur.
D.S: matematiksel değişiklerdir.

→ Axiomatic semantic:
state yok. prove etmek için gerekenlerdir.

variable: yer ayırma işleminin yapıldığı alanda tutulan veriye variable denir.
• location, • referencing, • scope, • lifetime, • type cheking, • Initialization, • Type compatibility'si vardır.

Six attribute:
- Name, - Address, - Value, - Type, - Lifetime, - Scope

Names
→ Comments, line boundaries ve white space name değildir.
→ Name constant: #define tanımlamaları. avantajları readability, modifiability,
→ Aliases: farklı değişkenlerin aynı memoriye refer etmesi

Variable initialization: → Başlangıç bindinginin yapılması

Static variables: Compile time'da memory cellerine bound edilir.
→ avantaj: efficiency, history sensitive
→ dezavantaj: less flexible

Stack dynamic variables: → stack'ten runtime'da allocate yapar. Execution'dan sonra ise deallocate yapar.
avantaj: recursiona izin verir. Lazım olduğunda ihtiyaç olunan yerde allocate edilir.
dezavantaj: overhead of allocation and deallocation indirect addressing

Explicit heap dynamic variable:
Explicit olarak allocate ve deallocate edilir. Sadece pointer ve reference üzerinden yapılır.
avantaj: Dinamik storage (flexibility)
dezavantaj: reliability azdır ve inefficienttir.

Implicit heap-dynamic variables
Value assign edildiğinde heap'te storage ayrılır.
Advantage: flexibility, writability
Disadvantage: Inefficient çünkü dynamictir. Hatayı bulmak zordur.

Primitive types :

Ambiguity ve bunun ortadan kaldırılması :

→ operatörlerin öncelikleri tanımlanırsa bu sorun ortadan kalkar.

Attribute Grammers :

Dilin yapısını Context Free Grammer'den daha iyi açıklayan bir araçtır. Attribute grammer CFG için iyi bir ektir.

Binding :

→ Dynamic Type Binding :
avantaj : esneklik
dezavantaj : yavaş
type check zor.

implicit heap dynamic

avantaj : esneklik
dezavantaj : hata fırlatma yetersizliği
yetersiz, bütün özellikler dinamik

Type checking :

→ Operatörün operand ile uygunluğunu kontrol eder.

→ Tip binding statik ise tip kontrolü statiktir.

→ dinamikse, dinamik olmak zorundadır.

→ Bir programlama dili bütün hataları fark ediyorsa, buna strongly type denir.

Referencing Environment

→ Bir komutun referans çevresi, o komut tarafından erişilebilen bütün isimlerdir.

Discriminated Union :
etiketlendirilmiş ortaklık

Static semanticler :

Tip uyumluluğu kuralları gibi kuralların BNF ile açıklanabilmesi zordur. Bu ve buna benzer durumlar static semantik kuralları ile açıklanırlar.

Dinamik Semantikler :

1) Operasyonel semantik → aksiyonlarla alakalıdır.
2) Denotasional Semantik → recursive function tabanlıdır.
3) Axiomatic semantik → state change yoktur. Programların doğruluğunu ispat için bir yöntemin geliştirilmesiyle ilgilidir.
→ durum değişimleri tanımlar.
→ onlar matematiksel fonksiyonlarla tanımlanır.

Stack Dynamic Binding

→ Tanımlandığı block aktif kaldığı sürece yaşar.
→ Bellek adresi hariç bütün özellikleri static olarak belirlenmiştir.

heap :

→ Sadece pointer ile erişilir.
→ avantaj : dinamik bellek yönetimi sağlar
→ Dezavantaj : yönetimi zor bu nedenle güvenilir değil

C ; type equivalence :

→ Normalde structural equivalence sadece struct ve union'da name equivalence

Names and scope :

Bir değişkenin scopu ; değişkenin görülebilir olduğu komutların alanıdır.

Data type :

• Programlama Dilleri •

Neden Programlama Dilleri dersi?
1) Dil öğrenmede yetkinliğimizin artması için
2) Belli bir dilin önemli özelliklerini anlayarak daha iyi kullanabilmek
3) Probleme uygun olan dilin seçimi
4) Hata ayıklarken özelliklerini bilmek faydalı olur.

Dil aileleri :
1) Imperative , 2) Concurrent , 3) O. Oriented , 4) Logic , 5) Functional

Binding =)
1) Compile Time'da yapılırsa : performans artar
2) Run Time'da yapılırsa : esneklik artar

***
* Borland JBuilder :
* Java için bütünleşmiş
* yazılım geliş. ortamı

Syntax analysis : tokenları belirler. (token is set of lexeme)

Programlama Dilleri Değerlendirme Kriterleri :

1) Okunabilirlik : Dilin okunabilir olması hata olasılığını azaltır ve bakımı kolaylaştırır.
- Overal simplicity , Orthogonality , structure type ...

2) Yazılabilirlik : - Dilin soyutlama yeteneği, dilin yazılabilirliğini önemli ölçüde etkilemektedir.
- Seçilen dilin eldeki probleme uygun olması işi kolaylaştırır.
- Overal simplicity , - Orthogonality , - Support for Abstraction

3) Güvenilirlik : Okunabilirlik , Yazılabilirlik , Type cheking and Exeption handling

4) Cost : ...

Lisp :
→ iki veri tipi var : atom ve list
→ Değişkenlere gerek yok.
→ Recursion ve koşullu ifadeler ile kontrol

Prolog :
→ Based on formal logic
→ Non procedural

Java :
→ doesn't include struct, union, enum
→ has reference but not pointers

2.pdf Türkçe
→ Bu derste programlama dillerinin syntax anlatmak için BNF adlı metadil kullanılacaktır.

* Bir programlama dilindeki en düşük düzeyli syntax birimlerine "lexeme" denir.

BNF : → terminal symols
→ non terminal symbols
→ unique start symbol
→ production rules

EBNF : BNF'in okunabilirliğini ve yazılabilirliğini artırmak amacıyla BNF'e bazı eklemeler yapılmış ve yenilenmiş BNF sürümlerine genişletilmiş BNF yada EBNf denir.
1) { } → istediğin sayıda seç, yada hiç seçme
2) | → veya
3) (|)

Names :
→ name length
→ case sensitive
→ Special words

Case sensitive : Dezavantajı : okunabilirlik

Abstract Data Type

10.3 slayt

1) Encapsulation : data tipi için tanımlanmış bütün operasyonların tek bir yerde bulunması

2) Information hiding : implementasyon detaylarını gizle.
hide the details.

? Algebric specification ?

- They don't specify
  - data representation
  - implementation details

- Object Oriented Programming -

Central concepts in object oriented languages
- Dynamic lookup - encapsulation - subtyping - inheritance

* C++ ; using static typing to eliminate search
* C++i problems with C++ multiple inheritance

- Object oriented program
  → Send messages to object

- Dynamic lookup
  Object → message (arguments)

In conventional programming
operation (operands)
meaning of operation is always the same
→ different code for different object

- Important distinction
  → Overloading is resolved at compile time
  → Dynamic lookup is a run time operation

- Encapsulation
  → kodu yazan kodun tamamını görüyorken, kullanıcı sadece belli bir kısmını görmektedir.

message → | object |

- Subtyping and inheritance
  → interface : external view of an object
  → subtyping : relation between interfaces
  → implementation : the internal representation of an object
  → inheritance : relation between implementation

Smalltalk language terminology

Object : instance of some class
Selector : name of a message
method : code used by a class to respond to message
Instance variable : Data stored in object

- Object Creation
  → To create an object, we copy an old one
  → We can add new methods, override existing ones, or even remove methods

→ Object - Oriented

- Subtyping
  inheritance mekanizmasına bağlanmıştır.

- encapsulation
  public, private, protected visibilty

** Overloading **
→ overloading is resolved at compile time
→ this is different from run-time lookup of virtual function

→ inheritance : fonksiyonlar override edilir.

- Encapsubtion in Smoltalk
  → methods are public
  → instance variable are hidden
→ subtyping : implicit, no static type system
→ inheritance : subclasses, self, super

- C++ Background :
  → Data abstraction
  → Objects and classes
  → Better static type cheking

* if you don't use a feature, your compiled code should be as efficient as if the language didn't include the feature.

Type Equivalence 2 type aynı olup olmadığına nasıl karar veririz?

→ Structural equivalence : 2 tipde aynı değeri içeriyorsa aynıdır. daha flexibilty'dir.

→ Name equivalence : same type names

→ Structural equivalence recursive type veya isim olduğunda complex olabiliyor. Bu yüzden name equivalence daha basittir.

- C, structural equivalence'dir. Sadece struct ve union'da name equivalance dir.
- Java name equivalence'dir.

* implicit conversion : upcasting
* explicit conversion : downcasting

* C++'da polymorphic fonksiyonlar "template"'ler.

- Side effects
  - → changes to memory, input / output
  - → Side efekt istenmeden olabilir
  - → Side efektsiz propran bir hiçtir.

* ternary operator : if expressiondır.

lazy evaluation : Side effect olmadığı zaman

→ Operation evaluated before the operand are evaluated

→ Operands evaluated only when necessary

Type checking vs Type inference

Standart type checking

```
int f(int x) { return x+1; };
int g(int y) { return f(y+1)*2 ; };
```

Type inference

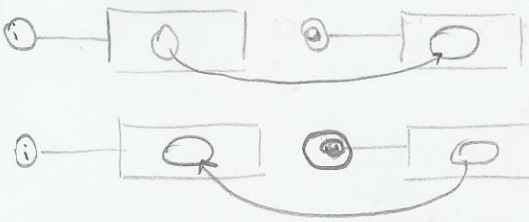
```
int f(x) { return x+1 }
int g(y) { return f(y+1)* 2; };
```

typelere bakmadan return valuesine göre type checking yapıyoruz.

* Dynamic scoping is not an accident
  - → User knows how to handle error
  - → Author of library function doesn't

Summary of Garbage Collector Technique
- Reference counting
  - * Directly keeps track of live cells
  - * Doesn't detect all garbage

Pass by value - Result



Pass by reference

→ formal parameters become "alias"

Pass by name

* Actual parameters only evaluated when they are needed
* Aynı parametre defalarca kez evaluated edilebilir.
* Lazy evaluation örneğidir.

→ Strict Evaluation vs Non-Strict Evaluation

→ fonksiyona kabul edilmeden önce parametreler evaluated edilir. (strict evaluation)

→ Sadece kullanılacağı zaman evaluated edilir (non-strict yani short-circuit & lazy evaluation)

Comparisons

→ Call by Value
- Efficient.
- Less flexible and less efficient without pointer

→ Call by reference
- Easiest to implement
- Explicit dereferencing

→ Call by value-result

→ Call by name:

difficult to implement
lazy evaluation

Funtional programming } ⇒ imperative programming:
- program = Data + Algorithms

OO programming:
- program = Object . message (object)

Functional programming:
- program = functions functions

Important features
- Everything is function
- No loop
- No variable or assignment (only constant values, arguments and returned values)
- No side effect

Eager Evaluation ⇒ order evaluation

→ functional style biroutk recursive benzer.

Advantages of fPL:
1) Simple semantic, concise, flexible
2) No side effect
3) Execution efficiency

≈ First-Order logic ≈

(boolean functions)

- Constant Symbol
  - Mary
  - 3
  - Green
- Function Symbol, - Predicate Symbol
  - father-of(Mary) = John
  - green(Grass)
  - greater(5,3)

- atomic sentence = which has value true or false
- complex sentence = connected by the logical connectives

Prolog

→ programming in logic

→ Program
- Axioms (facts): true statements

→ Functions
non-Boolean functions (successor(x))

→ Punctuation symbols
parentheses
comma
period

$(a \rightarrow b = a' \vee b)$

⇒ Logical programming = Logic + Control ⇒ Algorithm
(statement; prove goals from axioms)

Horn Clause

$b \leftarrow a_1$ and a_2 and a_3 and ---- and a_n.
(b: head; $a_1 \ldots a_n$: body)

$b \leftarrow$. fact
$\leftarrow b$. query

- First order logic

natural(0).
$\forall x$, natural(x) → natural(successor(x))

- Horn Clause

natural(0)
natural(successor(x)) ← natural(x).

* prolog syntax

:- For ←
, for and

~~⇒ Problem solving by resolution~~

⇒ Unification
Pattern matching to make statements identical.

→ Resolution and Unification
- program:
  - statements / facts (clauses)
- Goals:
  - Headless clauses, with a list of

* Possible Solutions to Name Clash

- Implicit : language resolves name conflicts with arbitrary rule
- Explicit : programmer must explicitly resolve name conflicts
- Disallow : name clashes : programs are not allowed to contain name clashes

Interfaces vs Multiple Inheritance

- C++ multiple inheritance :
  → Birden fazla class'dan extend edilmiş olunabilir.
- Java interfaces
  → Bir class birden fazla interface'den implement edilebilir.

Java : Javada efficiency ikinci plandadır.

→ Almost everything is an object
→ All object on heap, accessed through pointers
→ Java static typing yapar.
→ finalize() method garbage collectoru çağırır.
→ constructerda sen süper classı çağırmazsan otomatik olarak çağırılıyor.

* Subtyping is implicit, inheritance is explicit