Notes LP

Fernanda Guimarães

1 (05/08/19) What are programming languages?

Programming languages are turing complete. Html is not a programming language. Assembly is. The formal definition is:

- Syntax
- Semantics
-

Words are easier to remember than numbers.

1.1 Fortran

IBM. It brought two news:

- There are variables
- Control structures (loops, conditionals).

Parsing: read a chain of characters and transform it into a data structure (a tree).

1.2 Lisp

Paretheses.
Based on mathematical functions and lists.
News: no need for parsing, built on linked lists.

1.3 ALGOL

Two news:

- Type notation
- Begin and end

1.4 COBOL

Grace Hopper.
Looks like a natural language.

1.5 How many are there?

O'Reilly says that there are 2500, wikipedia says 650. Java is the most popular (portability).

1.5.1 Different purposes

- Fortan: scientific calculus
- Lisp: computer theory
- COBOL: comercial applications
- Algol: academic languages

1.6 C

Denis Reed
It was made to finish UNIX.
It's popular because the compiler already came with UNIX.

1.7 PHP

Recursive name.
Useful for web servers.
Came to supply the need for Pearl.
Came with Apache, not efficient.

2 (07/08/19) Types of languages

State = memory
Parsing = produce derivation trees for some chain of characters.
A program in x86 is a set of instructions.
Prolog isn't a patternized language.

2.1 Imperatives (state)

Turing machines.

- C
- Cpp
- Java
- Python
- C#

2.2 Declaratives (stateless)

There are no steps.

2.2.1 Functionals

Lambda calculus.

- ML
- Haskell
- Lisp
- Erlang

- Elixir
- Scala
- Clojure

2.2.2 Logicals

Horn clause.

- Prolog
- Datalog

2.3 Grammars

- Tokens (terminals)
- Non-terminals (variables)
- Production rules
- Start symbol

2.3.1 Types

- Regulars: super fast.
- Context-free: can only have a symbol on the left side of production.
- Context-sensitive: many symbols (right side is bigger or equal to left side).
- Irrestricted grammar: Turing Machines.

3 (12/08/19) Precedence

Parsing is used in compilers, valgrind, static verification, etc. There are two semantics aspects of languages:

- Associativity
- Precedence

In C, there are unary, binary and ternary operators. The closer to the roots, bigger the precedence. Attribution is associative to the right.

4 (14/08/19) Compilation

Search for: arithmetic identities of gcc.

Programming languages are usually compiled (ex assembly), virtualized (ex python) or interpreted (ex bash).

Virtualized are compiled to a virtual machine.

4.1 Why are some programs interpreted, others interpreted and others virtualized?

Because of efficiency. It's better to compile the program when the execution time is really large.

4.2 Compilation

The classical Sequence: [editor] –> source file –> [preprocessor] –> preprocessed source file –> [compiler] –> assembly language file –> [assembler] –> object file –> [linker] –> executable file –> [loader] –> running program in memory

5 (19/08/19) Introduction to ML

Important: an algoritm that in C has less complexity than in ML. There isn't implicit coersion. Everything is explicit.

Declarative Functional language. Follows the lambda-calculus. Program **is** a value, and not a sequence of state alterations. Every program in ML has a type. A bunch of functional languages have type inference.

Built around *unification*.

The five primitive types are: bool, int, real, char and string.

You can't compare real and int, but you can convert one to another.

Every if has an else, because every program is a value.

Functions have a very high precedence.

5.1 Tuples: cartesian product

Tuples are indexed by 1. There are no one-element tuples.

Every fun in ML receives only **one** parameter.

Type contructor = '*'. It's like a fun that receives types and returns types. It's like generics in Java and templates in cpp.

5.2 Lists

Read head and read tail in O(1), same types.

- [1,2,3];

val it = [1,2,3] : int list

- [1.0,2.0];

val it = [1.0,2.0] : real list

@ is O(n). :: is O(1) and associated to the right.

Explode splits a string into a list of chars.

If the '=' operator appears in a definition of a fun, then real numbers cannot be used.

You can force a fun to be real (it can come in several places):

*fun prod(a,b):real = a * b;*

The output of the type inference can be exponential.

6 (21/08/19) Pattern matching in ML

Undescores are better than a variable that is never going to be used. Two '' mean there are no real numbers.

7 (26/08/19) Type

Types are a set of values. Brainfuck and Forth don't have types. Type systems avoid some erros.

Advantages:

- Documentation

- Safety
- Efficiency
- Correctness

In C, the size depends of the compiler. R is a language for array manipulation, so is matlab and APL.

7.1 Primitive vs Constructed

Primitive is built-in. Constructed types are just sets built from other sets.
You can make constructed types by cartesian product, for example.
In C, an enumeration is a subset of ints. Structs are stored sequentially.
In Ml, you can only do a *comparison* with an enumeration.
The cardinality of a type is the product of its types cardinalities.

7.2 Vectors

Three abstractions: lists, vectors and strings.
Vectors are a multidimensional cartesian product of the same set (same type).

7.3 Union

The cardinality is the *sum* of the cardinality of its types. The space occupied is the largest element's size.

7.4 Functions

A map that maps the domain to a range. In C, you can pass a function as a parameters with a address.

7.5 Static vs Dyamic Typing

- Static examples: C, Java, *SML*, Cpp, Haskell.
- Dynamic examples: Python, Javascript, Lisp, PHP, Ruby.

  Static are more efficient. Bigger programs tend to be written in statically typed languages. Are more legible.

  Dynamically typed typed languages are more reusable. This kind of reuse is called duck typing.

7.6 Strongly vs weakly typed

A strongly typed language guarantees that a type will be always used as declared.

- Strongly: haskell, ml.
- Weakly: c, cpp. Ex: unions, coersion, idexing. Advantage: performance.

8 (28/08/19) Polymorfism

8.1 Strategies to discover types

- Implicit: types are inferred.
  - Inference: the compiler uses an algorithm that finds the correct type of each value. Examples: Haskell, Scala, SML.

  - Special names: In some old languages, the name of the variable gives away its type. Example: in old Fortran, integer variables should start with 'I'.

- Explicit: syntax determines types.
  - Annotations: the programmer must explicitly write the type of a symbol next to it. Examples: Java, C, C++.

8.2 Equivalence:

- name equivalence: two types are the same, if, and only if, they have the same name: C, Java, C++, etc. Advantage: legibility.
- structural equivalence: two types are the same if they have the same structure. Example: SML. Advantage: reusability.

8.3 Polymorfism

Python is way more reusable than C or SML. The secret to get closer to Python is polymorfism. A function or operator is **polymorphic** if it has at least two possible types.

Which statically typed language gets closer to python? Two types of polymorfism:

- ad-hoc (infinite symbols)
- universal

8.3.1 Ad-hoc

- Overload. Uses the types to choose the definition.
- Coersion. Uses the definition to choose a type conversion.

1. Overload: An overloaded function name or operator is one that has at least two definitions, all of different types. Many languages have overloaded operators. There are languages that allow the programmer to change the meaning of operators.
2. Coersion: A coercion is an implicit type conversion, supplied automatically even if the programmer leaves it out.

8.3.2 Universal

- Parametric
- Subtyping

1. Subtyping: Barbara Liskov's principle. Subtyping **isn't** the same as inheritance. Not the only mecanism to create subptypes.

9 (02/09/19) Lambda calculus

9.1 Lambda expressions:

Each lambda declares a different name.

```
<expr> ::= <name>
      | \lambda <name> . <expr> |
      | <expr> <expr>           |
```

$$(\lambda x \cdot x) \cdot w$$

, where

$$x_0 = \text{formal parameter}$$
$$x_1 = \text{function body}$$
$$w = \text{real parameters}$$

9.2 Numbers

A number is a function that takes a function s plus a constant z. The number N is formed by applying s N times on z.

```
Zero = \s.\z.z
One = \s.\z.sz
Two = \s.\z.s(sz)
Three = \s.\z.s(s(sz))
```

10 (04/09/19) Currying and Higher order

10.1 Currying

Then it is not possible to implement a function which take multiple parameters? Of course, it is possible, by a methodology called currying.In currying every function takes only one argument and returns a function. While the last function in this series will return the desired output.

10.2 Anonymous Functions

Starts with *fn* (lambda functions). You can't do a recursive anonymous function because it doesn't have a name. For you to do one, you need the *y combinator*.

In SML, we can create anonymous functions, e.g: (fn x => x + 2) 3.

'fn' is the equivalent of lambdas in SML, e.g: .x is equivalent to fn x => x

- val ZERO = fn s => fn z => z;
- val ONE = fn s => fn z => s z;
- val TWO = fn s => fn z => s (s z);
- val THREE = fn s => fn z => s (s (s z));

10.3 Higher Order

Each function has an order:

- A function that does not take other functions as parameters, and does not return a function value, has order 1
- A function that takes a function as a parameter or returns a function value has order n+1, where n is the order of its highest-order parameter or returned value

1. Foldr and Foldl Only return the same when operators are both associative and commutative.

11 (11/09/19) Data Types/algebric types

Slides In C, the union is the closest to algebric types.
Data constructors :: Labels Data Types :: constructors

11.1 Datatype: union in ML

New types can be defined using the keyword datatype.
These declarations define both: – type constructors for making new (possibly polymorphic) types – data constructors for making values of those new types
We've seen two data structures: lists and tuples:
Lists = nil | e * Lists
In C, enumerations are like:

```
#include <stdio.h>
enum Dia {Mon = 3, Tue, Wed, Thu, Fri, Sat, Sun};
int main () {
  printf("%d, %d, %d, %d, %d, %d, %d\n",
         Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  printf("%d\n", Mon + Tue);
}
```

In ML:

```
datatype day = Mon | Tue | Wed | Thu | Fri | Sat | Sun;
fun isWeekDay x = if not (x = Sat orelse x = Sun) then true else false ;
isWeekDay Mon;
```

11.2 Extint (of)

To recover a data constructor's parameters, use pattern matching. You can't do the same (pattern matching) with a function.

11.3 Option - type constructors

Is a polymorphic type. Used by predefined functions (or your own) when the result is not always defined

```
datatype 'a option = NONE | SOME of 'a;
```

12 (19/09/19) Review

- Ocaml and subtyping
- Types of languages
- Elixir and Erlang are on the same VM
- h::t
- Context-free: can only have a symbol on the left side of production and produce 0 or more things on the right side. Given a string, you can always say if a string belongs, but can't determine wether it's ambiguous (can generate an infinite set of strings).
  - Tokens (terminals)
  - Non-terminals (variables)
  - Production rules

  - Start symbol

- **Serious**: need to write grammar in prolog (attributes and no attributes). Ex: count number of constants.
- There are two semantics aspects of languages (for binary operators):
  - Associativity: which operator to evaluate first (right or left).
  - Precedence

In ml, are associative to the right.

- Turing Completes need loops.
- Compile, interpret, and virtualize.
- In Javascript, the browser gets the source code (no binary code). Not x86.
- Java is ASCII, bytecodes is a binary language executed in a hw called JVM.
- Scala, clojure, kotlin, groov: jvm.
- Tuples and lists are the only structures (actually tuples only).
- List operations
- The pogram with polynoms doesn't fall in the test
- The one with the triangles might
- six patterns in ml: variables, constants, lists, tuples, nil, h
- Types are a set of values. Brainfuck and Forth don't have types. Type systems avoid some erros.
- The cardinality of a type is the product of its types cardinalities.
- Vectors are a multidimensional cartesian product of the same set (same type).
- Tuples have different types
- In ml, the union is the datatype. The cardinality is the *sum* of the cardinality of its types. The space occupied is the largest element's size.
- Cartesians, Vectors, functions and unions.
- Static examples: C, Java, *SML*, Cpp, Haskell.
- Dynamic examples: Python, Javascript, Lisp, PHP, Ruby.
- Static vs Dyamic, Strongly vs Weak, Nominal vs Structurual, Inference vs annotation. Advantages of each one.
- Coersion is like converting.
- Advantage of weakly typed: performance.
- There's a spectrum of dynamic to static typing.
- There is a if then else in m.() in java, for example.
- Types of polymorfirsm
- Polymorfistm is about statically typed languages. It's point is to get statically typed to be more reusable.

- hd function in ml, e.g, is polymorfic.
- differnce in universal and ad-hoc.
- VAI CAIR OCAML e subtyping.
- Watch out with renaming in lambda-calculus (avoid variable captures).
- Operations with church number, you gotta know how to do. No succ declaration. Implement add given succ. Convert church number to int.
- Map, filter e fold.
- Scope is a region where a symbol exists.
- Almost every language is static scope.
- **Bash** is the only dynamic que presta.
- Two types of forming scope: blocks and namespaces.
- A tree can be a tree or 2 elements plus a tree.
-

13 (30/09/19) Memory allocation

Operating System delivers to the program all adressing space.
Memory is organized in three parts by compiler:

1. Static (Cobol/Fortran)
2. Stack (Algol)
3. Heap (Lisp)

Static has the same functionality in *C* and in *Java*:

$$\text{Static} == \text{compile time}$$

Consequence: only **one** (ALWAYS) instance of static variable, because its adress is known in compile time.

13.1 Static Memory

1. Static
2. Global
3. Program
4. Strings

Advantages: easy to program.
Disavantages: lose recursion (big) and [doubt].

14 Dynamic Memory

14.1 Stack

Program is usually in a contiguous memory area. Later there is an area that the compiler uses during the program execution, called stack. Usually in C, you only get into a function with a call and leave with a return. In python, Lua and ruby, there are yield and generators.

With a stack, I solve the recursion and cuncurrent allocation problem. Almost all languages have a stack that grows top-down.

14.2 Heap

Doesn't solve the OO problem nor the "good and evil". In *C*, the heap is used with a malloc. in Python, everywhere. In java, with *new*. In ML, *clojures* stay in heap as well. The data that live more than its functions live in the heap. In OOP, the heap is more used (data is more important than functions).

- **What is malloc's complexity?** free is always O(1). Malloc is O(1) if called alone. Malloc is online (non deterministic) if you call malloc and free alternatively.

Garbage collector is only in *heap*. There is no need for it in stack (just update a pointer). Heap: allocation and free is out of order.

Two ways to do garbage collection:

1. Counter - problems: cycles (functional languages don't have this problem)
2. Tracing (marcação e varredura): all imperative languages like Java. Mark the reachable space if memory is being used.

15 (07/10/19)

There is function and data (objects) orientation. Object O is a natural form to get encapsulation. Object orientation is the data knowing how to do stuff. Simula was the first OO language. Smalltalk was later (more know). Messages are methods today. It was not common to put operations in tables.

15.1 Language OO that don't have classes

- Lua
- Javascript

15.2 Feautres all OO

- have objects
- have methods
- heap
- subtyping
- Dynamic method invocation

15.3 Some features in OO (not common to all)

– Classes – Prototypes – Inheritance – Encapsulation – Polymorphism

15.4 What is an object?

Object is data. It is a table. There are pointers to functions and to itself.

15.5 Can you create obj without classes?

Yes. Example python and scala (clojures).

15.6 What is inheritance?

Reuse mechanism. In many lanuages like python and Java, inheritance is subtyping. Not in haskell and Ocaml.

15.7 Can you think about a situation in which we would like to have multiple inheritance?

Python and cpp have. Java doesn't. Few languages have multiple inheritance due to ambiguity.

15.8 What is the Liskov's substitution principle?

If S is a subtype of T, then S can be used in any situation where T is expected.

15.9 Methods in python and Java?

- In python, with inheritance, the call is proportional to the tree size.
- In Java, it is O(1).

15.10 Dynamic method invocation

- O(1) call is only possible in statically typed languages. For python, the type and adresses are only know in run time.

15.11 What is the complexity to find the target of a call in Python?

O(n).

15.12 What about the complexity to find the target of a call in Java?

O(1).

15.13 Top down vs bottom up

16 (09/10/19) Error handling

A type system limits the number of programs you can write. and proprietary types in rust.

16.1 Advantage of these in rust:

Avoid aliasing. Avoid errors.

16.2 Vectorization:

Use big registers to do matrix multiplication. Only works when x and y point to different places.

16.3 Even stronger type systems [idris/agola]

Dependent types. Disvantage: limit the number of programs one can write. Advantage: security.

16.4 C

5 types of error handling:

- Guard method outside func is bad bc you have to remember to put it everywhere you cal pop.
- Guard method inside func is bad bc you don't know if the stack is really empty.
- Option type: very good, but you receive a option. In swift everything is an option. Atoi returns a special value in c.
- Exit: lose reuse bc it aborts program.
- Errno: you can get the number of bits errors.
- Long jump: dynamic go to to simulate exception.

16.5 Eiffel (contract programming)

- 80's language
- pre conditions (require)
- pos conditions (ensure)

16.6 Exceptions (most common)

3 things:

1. define - class Ex(Exception)
2. raise - raise
3. treat - try except

Python (class that inherits Exception), Java, cpp, sml

16.6.1 What is an excption?

16.6.2 Advantages

16.6.3 What is an exception in Java?

*

17 (14/10/19) Parameters

- Formal parameters: in function declaration.
- Real parameters: in funcion call.
- Positional: match by position. Almost all programming languages.
- Nominal: match by name. Python, Ada.

  If you use the first argument as nominal and the second as positional, it doesn't compile. Optional parameters are also possible if you use default values. Cpp and python, for example, have optional parameters.

  Const in method determines it doesn't alter the state of the object.

  In C, stdarg allows you to use a variable number of parameters.

  printf accpets a variable number of parameters. It makes a parsing with %.

```
def f(a, L=[]):
  L.append(a)
  return L
```

If you do this and call f(3) after f(2), it returns a list [2, 3].

17.1 Strict

17.1.1 Pass by value

Java, python. If you pass something with pointers, it is by value (address copy) - syntatic sugar. In python, you can alter the thing which you are pointing to, not the thing itself.

17.1.2 Pass by reference

Cpp (**only one**). If you pass something with the address as formal parameters, it is by reference.

17.1.3 Not used anymore

- Result: the parameter is uninitialized, and it is filled with a value

before the function returns.

- Value result: same as result, but parameter is initialized.

17.1.4 Why are there only a few pass by reference languages?

Collateral effects. With attribution, there is a change in state. By altering the name, you can provoke a change in another name. If a program has pointers, it is really hard for the program to make the code faster (ex matrix multiplication).

17.1.5 How to make swape without an aux?

- XOR swap. If you try to swap &a with &a in cpp, everything is 0. Bad business (same address). XOR of something with itself is 0. If you alter one, the effect will be felt on the other thing.

17.1.6 Keyword restrict (c only - not cpp)

17.2 Strict vs Lazy languages:

Strict: arguments are evaluated before passing. Python.

- False and 1 / 0 == 0
- False
- def myand(a, b): return a and b
- myand(False, 1/0 == 0)
- ZeroDivisionError: integer division or modulo by zero

17.3 Lazy or not strict

- Haskell, Scala, cpp and c. Only evaluate if you actually need it.

17.3.1 Pass by macro

1. C
2. Cpp
3. Lisp
4. Rust

Macro expansion: the parameter is evaluated every time it is used.

1. 10.0) What is the advantage of macro expansion? Resource that allows programmers to alter language syntax, pass type as parameters. Ex foreach. With macros, u can create parametric polymorphism in c.
2. 10.1) What is the problem with macro expansion (serious problem)? Two problems: variable capture and multiple evaluation (++b).

```c
#include "stdio.h"
#define SWAP(X,Y) {int temp=X; X=Y; Y=temp;}
int main() {
  int a = 2;
  int temp = 17;
  printf("%d, temp = %d\n", a, temp);
  SWAP(a, temp);
  {int temp=temp; temp=B; temp=temp;}
  printf("%d, temp = %d\n", a, temp);
}
```

17.3.2 Pass by name (algo/simula)

NÃO CAI NA PROVA.

18 (16/10/19) Introduction to prolog

- Declarative: stateless.
- Logical language.
- Order **matters**.
- Horn clause (Alfred Horn).
- Doesn't have a pattern (each compiler has its own version).
- REPL: interactive environment (java doesn't have it) - ipython.
- Programs have

18.1 Horn Clauses are implications

There is a procedure to check if a clause is false or true.

```
~a and ~b and ~c -> y or u
```

Easiest language will see.

18.2 Unification

- Alan Robinson
- Same algorithm to make type inference in ml, haskell, etc. The cpp auto is not the same, it is much simpler.
- Exponential.

18.3 The core of prolog is the term:

- atoms:
  - constants: 123, -123, etc
  - alpha-numeric symbols: fernando, *, ., =, [.
  - atoms start with lower case.
- variable: names beginning with an upper-case letter: X, Fernando, _
- composite terms: an atom plus a sequence of terms between parentheses: x(Y, Z), x(y, Z)

```
atom (atom or composite terms)
```

18.4 Ex prolog program

- Predicates end with a dot. It has six truths (predicates).

```
parent(margaret, kim).
parent(margaret, kent).
parent(herbert, jean).
```

18.5 How to discover if esther has a grand-grandchild?

```
?- parent(esther, C), parent(C, GC), parent(GC, GGC).
```

18.6 Inference rules(implications)

- Look like functions syntatically, but not semantically
- If the right side is true, the left is true.

```
greatgrandparent(GGP, GGC) :- parent(GGP, GP), parent(GP, P), parent(P, GGC).
```

- Ancestor

```
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- ancestor(parent(Z, X), Y).
```

18.7 Lists

- Pattern matching

| is like in ml.

- Append:

```
myAppend([], L, L).
myAppend([H|L1], L2, [H|L3]) :- myAppend(L1, L2, L3).
```

- Member

```
myMember(H, [H|_]).
myMember(H, [_|T]) :- myMember(H, T).
```

- Select

```
mySelect(H, [H|L1], L1).
mySelect(X, [H|L], [H|LR]) :- mySelect(X, L, LR).
```

- Rev(L1, L2)

```
rev([], []).
rev([X|T], R) :- rev(T, RR), myAppend(RR, H, R).
```

- Siblings: prolog has denial, but it doesn't work well.

- Sublist:

```
sublist([], []).
sublist([H|T], [H|S]) :- sublist(T, S).
sublist([_|T], S) :- sublist(T, S).
```

- Perm:

```
perm([], []);
perm([H|T], L) :- perm(T, Lx), select(H, L, Lx).
```

18.8 Symbolic manipulations