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
- Erlang
- Elixir
- Scala

2.2.2 Logicals

Horn clause.

- Prolog
- Datalog

2.3 Grammars

- Tokens (terminals)
- Non-terminals
- 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 At tribution 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

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.

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

$$\begin{aligned} x_0 &= \text{formal parameter} \\ x_1 &= \text{function body} \\ w &= \text{real parameters} \end{aligned}$$

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.
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.