MI-GLSD-M1 -UEM213 : Programming paradigms

PW(TP): The research language OZ

A. HARICHE

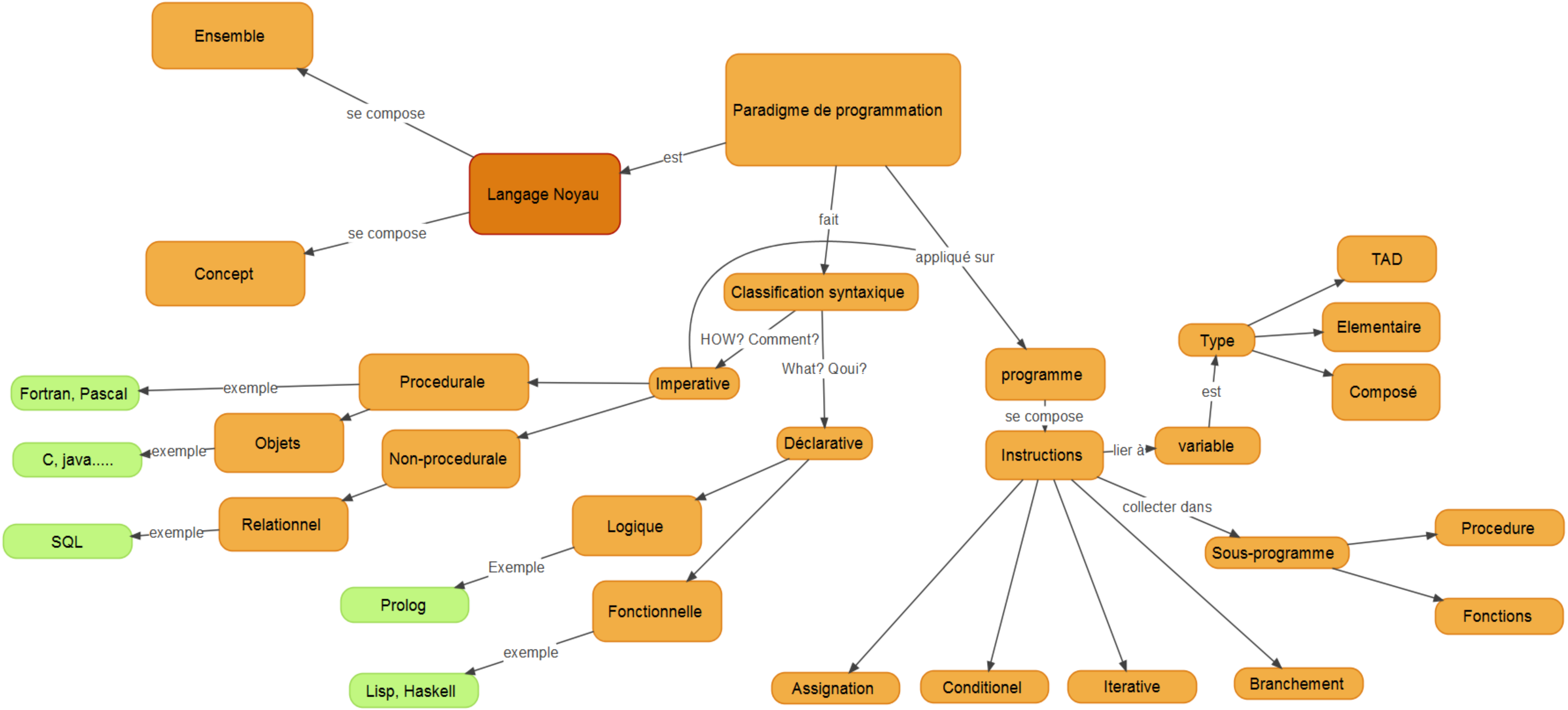
University of Djilali Bounaama, Khemis Meliana (UDBKM)

Faculty of sciences & technology

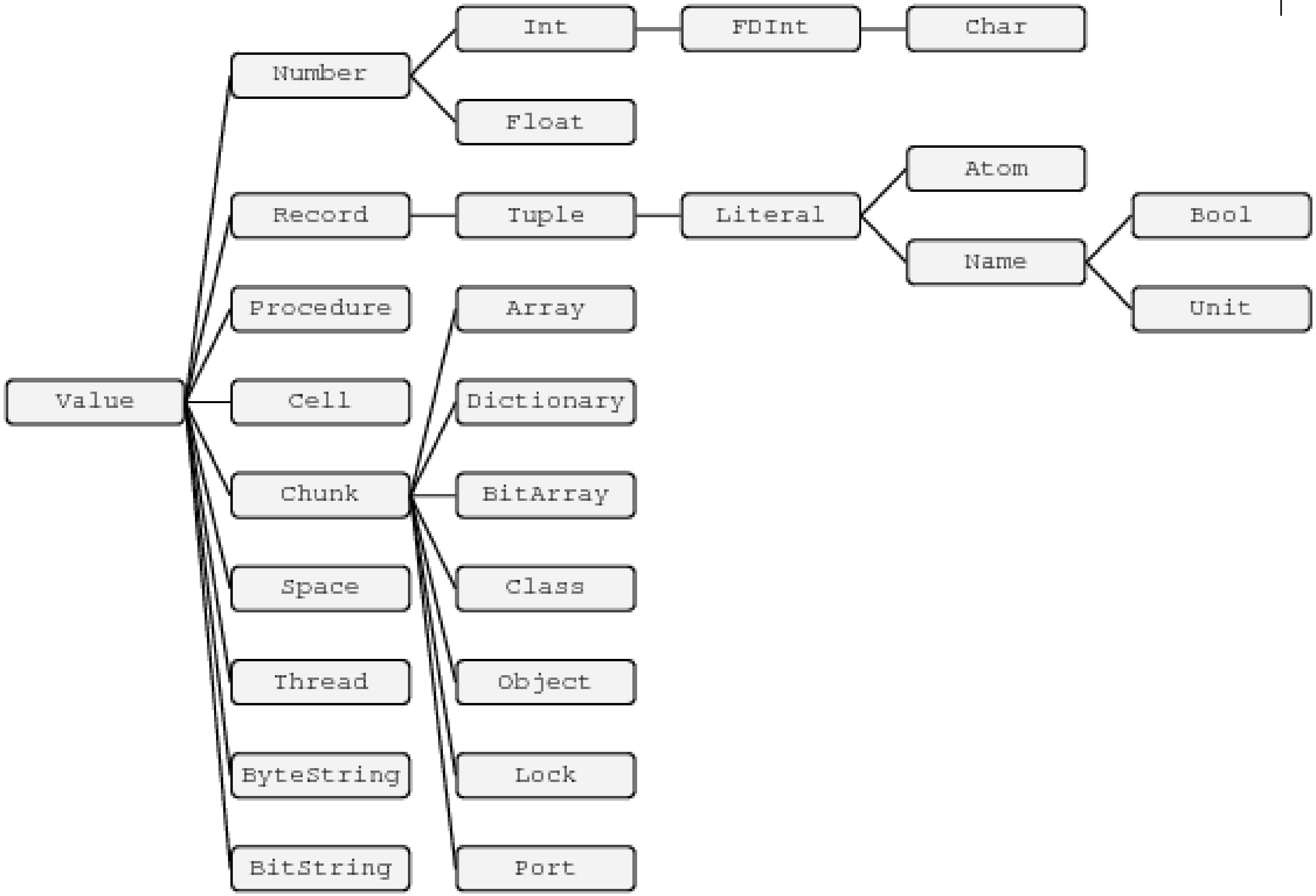
Mathematics & computer sciences department

a.hariche@univ-dbkm.dz

In the last lectures



OZ Conceptual types



OZ types

Abbreviation	Type
A	atom
B	bool
C	chunk
F	float
I	integer
K	class
L	literal
N	name
O	object
P	procedure
R	record

OZ types

Abbreviation	Type
S	string
T	tuple
U	unit
V	virtual string
X Y Z	value
FI	number
LI	feature
AFI	atom, float, or int
PO	unary procedure or object
Xs	lists of elements of type *X*

Primary types

Numbers are either integers or floats.
Literals are either atoms or names.
Tuples are special records whose features are the integers from 1 to n for some integer n, n >= 0.
Procedures are classified according to their arity. We speak about n-ary procedures.
Chunks serve to represent abstract data structures. They are defined similarly to records but provide only a restricted set of operations. This makes it possible to hide some or all of their features. Typical chunks are objects and classes, locks and ports, and arrays and dictionaries. There are chunks which do not belong to these types.

OZ as research language

- Research language with a wide range of programming/system abstractions to develop quickly and robustly advanced applications. And, yet it is a simple and coherent design.

- It is a high level programming language that is designed for modern advanced, concurrent, intelligent, net worked, soft real-time, parallel, interactive and pro-active applications.

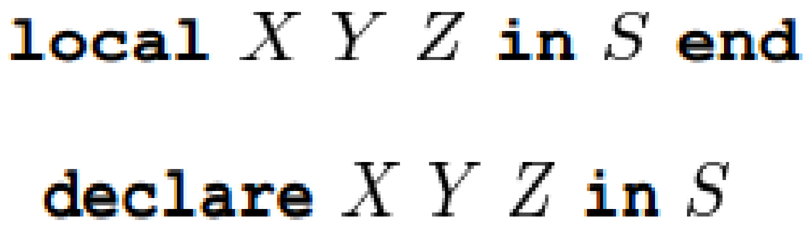
OZ with multiple paradigms

Oz combines :

- Oriented Object Programming: state, abstract data, types, classes, objects and inheritance.
- Functional Programming : it is providing:
  - ***first class procedures*** *:* procedures, threads, classes, methods, and objects.
  - ***Lexical scoping*** with privates calls during the compiling phase.
- Logic and constraint programming: logical variables, disjunctions, flexible search mechanisms and constraint programming.
- Concurrent language dynamical interactions of any number of sequential threads (data-flow threads) respecting a real data flow dependencies on the variables involved in each statement.

The sequential programming style of Oz

- Variables Declaration



local $X\ Y\ Z$ **in** S **end**

declare $X\ Y\ Z$ **in** S

The sequential programming style of Oz

- Variables assignement

Example : Skip

Skip. The statement skip is the empty statement.

Example : sequential execution?

S1 S2

Thread executes statements in a sequential order. However a thread, contrary to conventional languages, may suspend in some statement, so above a thread has to complete execution of S1 , before starting S2.

The sequential programming style of Oz

- Variables assignement
  - **Example : Numbers**

```
local I F C in
I = 5
F = 5.5
C = &t
{Browse [I F C]}
end
```

no automatic type conversion, So

5.0 = 5

will raise an exception

The sequential programming style of Oz

- Variables assignement
  - **Example : Lists**

```
local L1 L2 L3 Head Tail in
L1 = Head|Tail
Head = 1
Tail = 2|nil
L2 = [1 2]
{Browse L1==L2}
L3 = ´ | ´ (1:1 2: ´ | ´ (2 nil))
{Browse L1==L3}
end
```

The sequential programming style of Oz

- Conditional instruction
  - **If condition**

```
local X Y F Z in
X = 5
Y = 10
F = X > Y
if F = true then
Z = X
else
Z = Y
end
end
```

The sequential programming style of Oz

- Conditional instruction
  - **Case condition**

```
local X Y Z in
X = 5
Y = 10
case X >= Y then Z = X
else Z = Y end
end
```

The sequential programming style of Oz

- Sub-program
  - **Procedure**

```
local
Max = proc {$ X Y Z}
case X >= Y then
Z = X
else Z = Y end
end
X = 5
Y = 10
Z
in
{Max X Y Z} {Browse Z}
end
```

```
local Max X Y Z in
proc {Max X Y Z}
case X >= Y then
Z = X
else Z = Y
end
end
X = 5
Y = 10
{Max X Y Z} {Browse Z}
end
```

The sequential programming style of Oz

- iterative instruction
  - **For as procedure**

```
local
proc {HelpPlus C To Step P}
case C=<To then {P C} {HelpPlus C+Step To Step P}
else skip end
end
proc {HelpMinus C To Step P}
case C>=To then {P C} {HelpMinus C+Step To Step P}
else skip end
end
in
proc {For From To Step P}
case Step>0 then {HelpPlus From To Step P}
else {HelpMinus From To Step P} end
end
end
```

The sequential programming style of Oz

- Sub-program
  - **Function**

```
declare
fun {Map Xs F}
case Xs
of nil then nil
[] X|Xr then {F X}|{Map Xr F}
end
end
{Browse {Map [1 2 3 4] fun {$ X} X*X end}}
```

The sequential programming style of Oz

●● Sub-program **When To Function or not to Function.**

●● Use function definitions when things are really functional, i.e. there is *one output* and, possibly many inputs, and the output is a mathematical function of the input arguments.

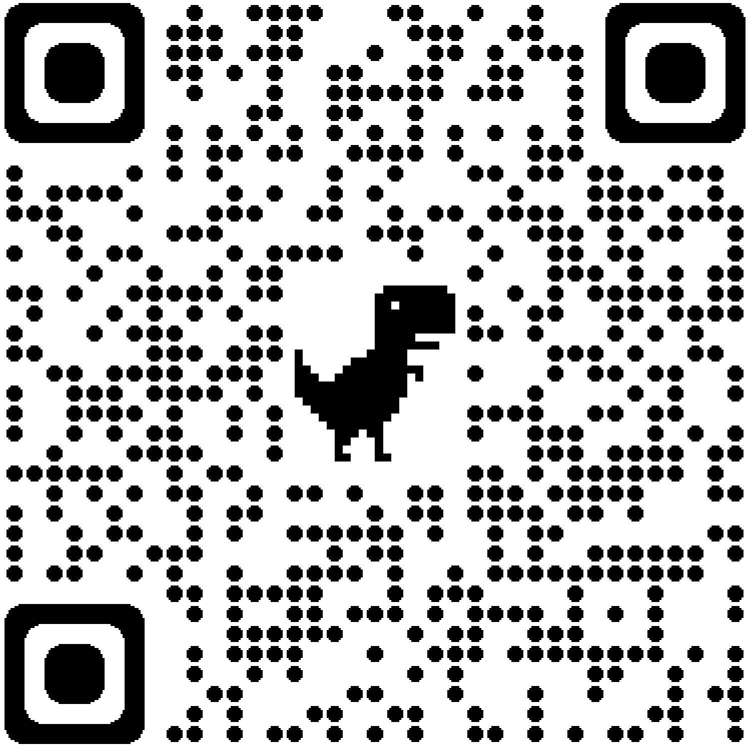
●● Use procedures in most of *the other cases*, i.e. multiple outputs or nonfunctional definition due to stateful data types or nondeterministic definitions

References

OZ syntax.

The tutorial of oz by seif Haridi

PLP_Drive Space



https://drive.google.com/drive/folders/1YBCIZzAldeiT19DIfDiREQwP-NAQ1qMN

Many important ideas

Louv1.1x

- Identifiers and environments
- Functional programming
- Recursion
- Invariant programming
- Lists, trees, and records
- Symbolic programming
- Instantiation
- Genericity
- Higher-order programming
- Complexity and Big-O notation
- Moore’s Law
- NP and NP-complete problems
- Kernel languages
- Abstract machines
- Mathematical semantics

Louv1.2x

- Explicit state
- Data abstraction
- Abstract data types and objects
- Polymorphism
- Inheritance
- Multiple inheritance
- Object-oriented programming
- Exception handling
- Concurrency
- Nondeterminism
- Scheduling and fairness
- Dataflow synchronization
- Deterministic dataflow
- Agents and streams
- Multi-agent programming