



# Mass parsing

Mariano Ceccato

ITC-Irst

Centro per la ricerca  
Scientifica e Tecnologica

[ceccato@itc.it](mailto:ceccato@itc.it)

# Problem

- ◆ TXL takes only one file as input and only one file as output
- ◆ In the real world, reverse engineering problems dealt with many source files

```
> txl -o class_diagram.dotty test.java extract_diagram.Txl
```



# Improper solution

Join all the source files in one huge file

- ◆ Not always possible, because for some languages source file names have semantics (e.g. Java)
- ◆ When possible it causes loss of the information about the actual source location, loss of modularity.

```
class A{ int x, y; }  
class B { }  
class C extends B{ A a; }
```

test.java

# Proper solution

- ◆ The same TXL program accesses several source files
- ◆ It parses and analyzes each of them
- ◆ It prints a result concerning all of them

```
class A{ int x, y; }
```

A.java

```
class B { }
```

B.java

```
class C extends B{ A a; }
```

C.java

# Mass parsing

Mass parsing:

- ◆ Store in a text file a list containing the names of the files to parse.
- ◆ The program reads this text file and then it builds a list
- ◆ For each name in the list:
  - Read the source file associated,
  - Parse it generate the AST,
  - Analyze the code and store data
- ◆ Join the data collected, print and exit

# Text file

- ◆ It is a file containing stringlit
- ◆ Each stringlit contains the path for a file to parse
- ◆ Stringlit are delimited by “   and   ”

```
"/home/ceccato/A.java"  
"/home/ceccato/B.java"  
"/home/ceccato/C.java"
```

list.txt

# The read built-in function

- ◆ The source file name is the argument of the call.
- ◆ The source file is parsed and a parse tree is returned.
- ◆ The root for the new tree shares the same type of the subject of the call.

```
function read_from_file
  ..
  construct a_list [repeat stringlit]
    _ [read "list.txt"]
  ..
end function
```

# EACH built-in

- ◆ A rule/function accepts a parameter of type **[T]**
- ◆ A list of type **[repeat T]**
- ◆ “**each**” produces a distinguished call for each entry in the list

```
construct accumulator [number]
  zero [+ entry_n1]
  [+ entry_n2]
  [+ entry_n3]
  ...
```

```
construct entry_list [repeat number]
  entry_n1 entry_n2 entry_n3 ...
construct accumulator [number]
  zero [+ each entry_list]
```

# Parse each entry in the list

- ◆ Extract the name from the list
- ◆ Using the **read** built-in function read the file and parse it as a *[program]*
- ◆ Extract data from it and collect them

```
function analyze_one_file the_file_name [stringlit]
  ...
  construct the_code [program]
    old_prog [read the_file_name]
  construct new_info [repeat dotty_edge]
    old_info [extract_edges_from the_code]
  ...
end function
```

# Mass parsing

Mass parsing:

- ◆ A text file holds a list containing the names of the files to parse.
- ◆ The program reads this text file and then it builds a list
- ◆ For each name in the list:
  - Read the source file associated,
  - Parse it generate the AST,
  - Analyze the code and store data
- ◆ Join the data collected, print and exit