



# The TXL Programming Language

Filippo Ricca & Mariano Ceccato

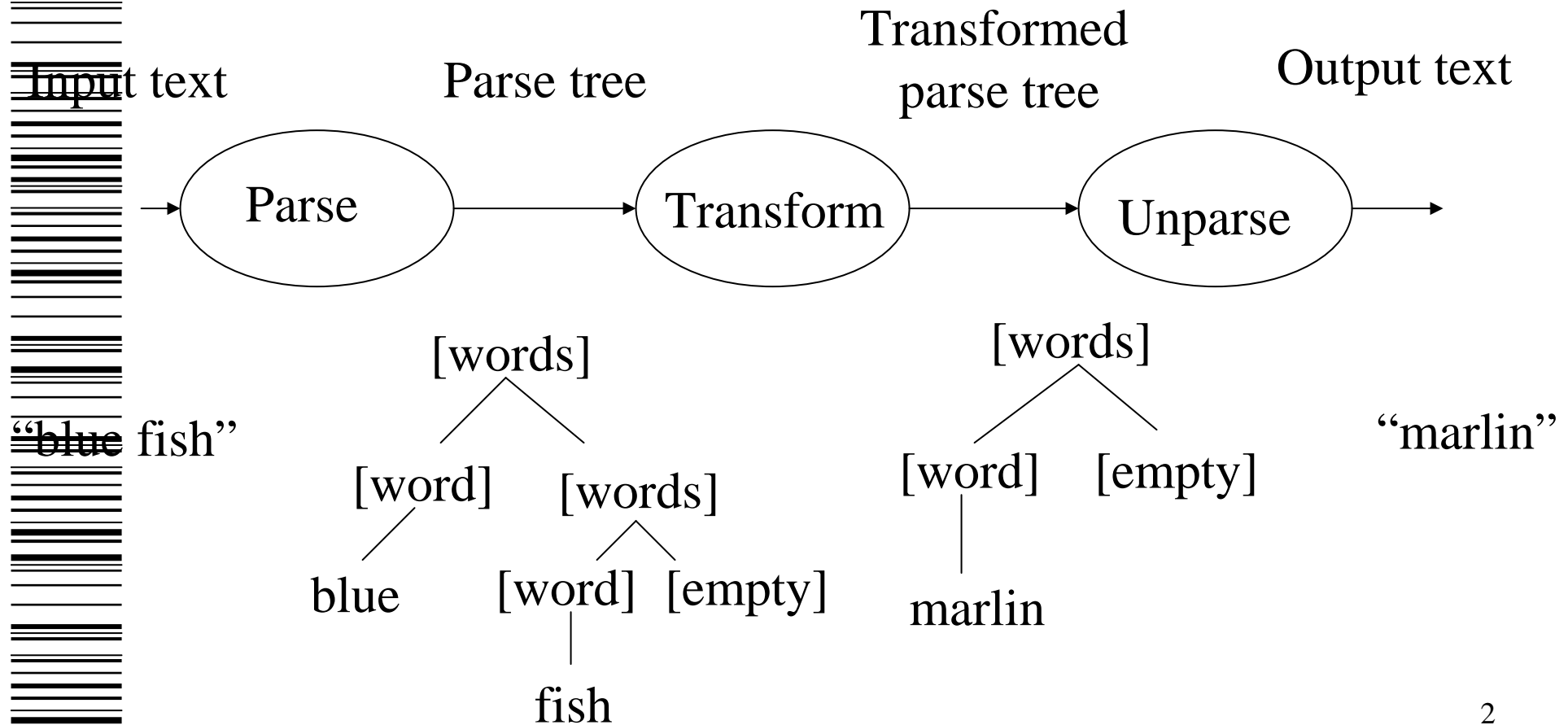
ITC-Irst

Istituto per la ricerca Scientifica e  
Tecnologica

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

# The three phases of TXL

txl “input file” “txl file”





---

# Anatomy of a TXL program

---

◆ Base grammar

The base grammar defines the lexical forms (**tokens** or **terminals**) and the syntactic forms (**non-terminals**).

◆ Grammar overrides

The optional grammar overrides non-terminal of the base grammar.

◆ Transformation rules

The ruleset defines the set of transformation rules and functions

# Anatomy of a TXL program

Example:

- ◆ Base Grammar
- ◆ Grammar overrides
- ◆ Transformation rules

Expr grammar

```
include "Expr.Grammar"  
redefine expr  
    ...  
| exp([number], [number])
```

```
include "Expr-exp.Grammar"  
rule main  
rule one  
rule two
```

# Specifying Lexical Forms

- ◆ Lexical forms specify how the input is partitioned into tokens.
- ◆ Predefined defaults include identifiers [id] (e.g. ABC, rt789), integer and float [number] (e.g. 123, 123.23, 3e22), string [string] (e.g. “hi there”).
- ◆ The **tokens statement** gives regular expressions for each class of token in the input language.

Example:

```
tokens
```

```
    hexnumber
```

```
    “0[xX][\dABCDEFabcdef]+”
```

```
end tokens
```

# Specifying lexical Forms (cont'd)

**tokens**

name "regular expression"

**end tokens**

Regular expression:

- ◆ Any single char (not [, ]) not preceded by a \ or # simply represents itself.
- ◆ Single char patterns: ex. \d (digits), \a (alphabetic char).
- ◆ Regular expression operators: [PQR] (any one of), (PQR) (sequence of), P\*, P+, P?.

# Specifying lexical Forms (cont'd)

## **keys**

procedure repeat 'program

## **end keys**

## **compounds**

:= >= <=

## **end compounds**

## **comments**

/\* \*/

## **end comments**

- ◆ The **keys** specifies that certain identifiers are to be treated as unique special symbols.
- ◆ The **compounds** specifies char sequences to be treated as a single terminal.
- ◆ The **comments** specifies the commenting conventions of the input language. By default comments are ignored by TXL.

# Specifying Syntactic Forms

- ◆ The general form of a **non-terminal** is:

**define** name

alternative1 | alternative2 ... | alternativeN

**end define**

- ◆ Where each **alternative** is any sequence of terminal and non terminal (N.B: *enclosed in square brackets*).
- ◆ The special type [program] describes the structure of the entire input.

# Specifying Syntactic Forms (cont'd)

- ◆ Extended BNF-like sequence notation:

[repeat x]      sequence of zero or more ( $X^*$ )

[list X]        comma-separated list

[opt X]         optional (zero or one)

... are equivalent

```
define statements  
  [repeat statement+]  
end define
```

```
define statements  
  [statement]  
  | [statement] [statements]  
end define
```

# Specifying Syntactic Forms (cont'd)

**key**

procedure begin 'end int bool

**end key**

**define** proc

procedure [id] [formalParameters]

'begin

[body]

'end

**end define**

**define** formalParameters

'([list formalParameter+])

| [empty]

**end define**

**define** formalParameter

[id] ': [type]

**end define**

**define** type

'int | 'bool

**end define**

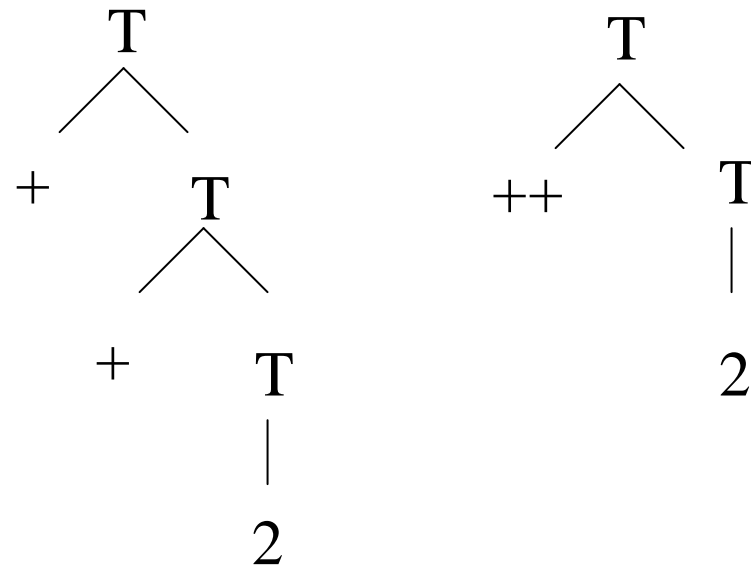
# Ambiguity

◆ TXL resolves ambiguities by choosing the first alternative of each non-terminal that can match the input.

Example: T-language

```
define T
  [number]
  | ([T])
  | + [T]
  | + + [T]
end define
```

++2





# Transformation rules

---

---

- ◆ TXL has two kinds of transformation rules, **rules** and **functions**, which are distinguished by whether they should transform only one (for functions) or many (for rules) occurrences of their pattern.
- ◆ **Rules** search their scope for the first instance of their target type matching their pattern, transform it, and then reapply to the entire scope until no more matches are found.
- ◆ **Functions** do not search, but attempt to match only their entire scope to their pattern, transforming it if it matches.

# Rules and functions

```
function 2To42
```

```
  replace [number]
```

```
    2
```

```
  by
```

```
    42
```

```
end function
```

```
rule 2To42
```

```
  replace [number]
```

```
    2
```

```
  by
```

```
    42
```

```
end rule
```

```
2 ----> 42
```

```
3
```

```
2 6 2 78 4 2
```

Rules search the pattern!

```
2 ----> 42
```

```
3
```

```
2 6 2 78 4 2 ----> 42 6 42 78 4 42
```

# Searching functions

```
function 2To42
  replace * [number]
    2
  by
    42
end function
```

2 ----> 42

3

2 6 2 78 4 2 ----> 42 6 2 78 4 2

Note: change only \*

# Syntax of rules and functions

Simplified and given in TXL.

```
'rule [ruleid] [repeat formalArgument]
  [repeat construct_deconstruct_where]
'replace [type]
  [pattern]
  [repeat construct_deconstruct_where]
'by
  [replacement]
'end rule
```

The same for functions!

**N.B.** If the 'where-condition' is false  
the rule can not be applied and  
the result is the input-AST.

# Built-in functions

```
rule resolveAdd
  replace [expr]
    N1 [number] + N2 [number]
  by
    N1 [add N2]
end rule
```

```
function add
  ...
end function
```

```
rule resolveAdd
  replace [expr]
    N1 [number] + N2 [number]
  by
    N1 [+ N2]
end rule
```

... are equivalent!

# Built-in functions (cont'd)

rule sort

replace [repeat number]

N1 [number] N2 [number] Rest [repeat number]

where

N1 [ $>$  N2]

by

N2 N1 Rest

end rule

22 4 2 15 1 -----> .... -----> 1 2 4 15 22