The TXL Programming Language

Filippo Ricca
ITC-Irst
Istituto per la ricerca
Scientifica e Tecnologica
ricca@itc.it

Recursive functions

```
function fact
   replace [number]
      n [number]
   construct n_minus_one [number]
      n [- 1]
   where n [> 1]
   construct fact_minus_one [number]
      n_minus_one [fact]          <-- Recursion
   by
      n [* fact_minus_one]
end function
```

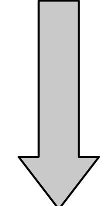
Using rule parameters

```
rule resolveConstants
    replace [repeat statement]
      'const C [id] = V [expr] RestOfscope [repeat statement]
    by
      RestOfScope [replaceByValue C V]
end rule

rule replaceByValue ConstName [id] Value [expr]
    replace [primary]
        ConstName
    by
        (Value)
end rule
```

⟵ Parameters

Example:

Const Pi = 3.14;
Area := r*r*Pi;



Area := r*r*3.14;

Deconstruct and searching functions

```
rule vectorizeScalarAssignments
   replace [repeat statement]
      C1 [statement] C2 [statement] rest [repeat statement]
   deconstruct C1
       V1 [var] := E1 [expr];
   deconstruct C2
        V2 [var] := E2 [expr];
   where not
         E2 [reference V1]
    where not
          E1 [reference V2]
    construct Passign [statement]
          <V1, V2> := <E1, E2>;
    by   Passign rest
end rule
```

Example:

```
x:=x+1;
y:=t+4;
```

→ `<x, y>:= <x+1, t+4>;`

```
x:=x+1;
y:=x+4;
```

No!

```
function reference V [variable]
    match * [variable]
         V
end function
```

Working with Global Variables

- **Global variables** are a rich and powerful feature that can be used for many distinct purposes, including:

  - **global tables**.

  - **multiple results from a rule**.

  - **“deep” parameters**.

  - **“message-passing” communication between rules in a rule set (e.g, avoiding interference).**

Setting Global Table

- **Global tables** can be **set up** using an export clause before the replace clause in the main rule of a program.

Example:

```
function main
    export Table [repeat table_entry]
        "Veggie" -> "Cabbage"
        "Fruit" -> "Apple"
        "Fruit" -> "Orange"
    replace [program]
        P [program]
    by
        P [R1]
end function
```

```
define table_entry
   [stringlit] -> [stringlit]
end define
```

Adding Table Entry

- **Global tables** can be **modified** by exporting a new binding for the table based on the imported original binding.

Example:

```
function addTableEntry
    import Table [repeat table_entry]
    …
   construct newEntry [table_entry]
              …
   export Table
         Table [. NewEntry]
   …
end function
```

Searching in a Table

- Global tables can be easily **queried** using **searching deconstructors**.

Example:

deconstruct * [table_entry] Table
Kind [stringlit] -> “Orange”

- The binding for “Kind” will be the [stringlit] “Fruit”. If no match were to be found, then the deconstructor would fail.

Avoiding interference between rules

```
function shiftByOne
    export Flag [id]
        'not_found
    replace [number]
        N [number]
    by
        N [replaceOneByTwo] [replaceTwoByThree]
end function
```

We want: 1 ---> 2 ~~--->~~ 3

2 ---> 3

```
function replaceOneByTwo
    replace [number]
        1
    export Flag
        'found
    by 2
end function
```

```
function replaceTwoByThree
    import Flag [id]
    deconstruct Flag
        'not_found
    replace [number]
        2
    by
        3
end function
```

Counting items in TXL

- TXL can be used for counting items (i.e. LOCs, number of cycles, etc.).

For example: given a tag-language counting the number of tags and end-tags.

```
<a>
    uno
        <b> due </b>
```

tags: 2
end Tags: 1

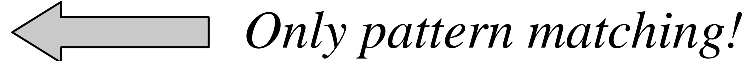
```
% Tags grammar

define program
   [repeat element]
end define

define element
    [Tag] | [endTag] | [id]
end define

define Tag
    < [id] >
end define

define endTag
    </ [id] >
end define
```

```
% Count number of tag

function main
   replace [program]
         P [program]
   construct ListTags [repeat Tag]
     _ [^ P]
   construct NumberTags [number]
     _ [countTag each ListTags] [printf]
   by
end function
```

R1 [^ X1]

Replace R1 of type [repeat T] with a sequence consisting of every subtree of type [T] contained in X1.

```
function countTag A [Tag]
    replace [number]
        N [number]
    by
        N [+ 1]
end function

function printf
    match [number]
        N [number]
    construct PrintObj [number]
        N [print]
end function
```



Only pattern matching!

print is a built-in function!

Using attributes

TXL allows a grammar definition to have attributes associated with it:

1. **Attributes** act like optional non-terminals, but normally do not appear in the output (**txl –attr** force the print of all attributes).
2. **Attributes** may be added to the parse tree during transformations.
3. **Attributes** are denotated in the grammar by the nonterminal modifier attr.

```
define type
      'int | 'string
end define

define typed_id
     [id] [attr type]
end define

function InferType expr [expression]
      replace [typed_id]
           Id [id]
      deconstruct expr
           f  [number] op [operator] s [number]
       by
            Id 'int
end function
```

The attribute 'type' is Optional.

Remark

1. Several functions (or rules) may be applied to a scope in succession. For example:

X [f][g][h]   (the meaning is: h(g(f(X))))

Exercises

- Adding to the 'expr-grammar' the exponential i.e Exp(x, n). Computing the exponential:

  - in syntax way: ex. Exp(2, 3) ----> 2*2*2

  - in semantic way: by means a recursive function that substitute at Exp(x, n) the correct value.

1+exp(3, 2) ⇒ 1+3*3 ⇒ 1+9 ⇒ 10

1+exp(3, 2) ⇒ 1+9 ⇒ 10

Homework

- Implementing a simple version of "commands-language" where commands can be:

  - assignments i.e. [id] := [expr];

  - declarations i.e. const [id] = [number];

- Implementing some transformation rules (page 19) that substitute in the assignments identifiers with related values.

Example:

Const Pi = 3.14;
Area := r*r*Pi;

→

Area := r*r*3.14;