Evolving TXL

Adrian Thurston and James Cordy

School of Computing
Queen's University
Kingston, Canada

TXL Overview

- Rule-based programming language for source transformation
- Function programming semantics overlaid on top for control
- Originally designed for prototyping modest syntactic enhancements
- Now used for much larger projects

TXL Overview

- LS/2000 – Legacy software maintenance system
- Document, math and table recognition
- Database and semantic web
- Security analysis and formal verification

A Need to Grow TXL

- Growth both in application size and domain
- TXL shows signs of strain
  - Software engineering perspective
  - Transformational programming perspective
- Example-based approach

Selection in TXL

```
function findApply SearchKey [number]
    replace [node]
        Node [any]
    by
        Node
            [findLeft SearchKey]
            [findRight SearchKey]
            [findHere SearchKey]
end function

function findLeft SearchKey [number]
    replace [node]
        Key [number] Val [value]
        Left [node] Right [node]
    where
        SearchKey [< Key]
    by
        Key Val
        Left [findApply SearchKey] Right
end function
```

```
function findRight SearchKey [number]
    replace [node]
        Key [number] Val [value]
        Left [node] Right [node]
    where
        SearchKey [> Key]
    by
        Key Val
        Left Right [findApply SearchKey]
end function

function findHere SearchKey [number]
    replace [node]
        Key [number] Val [value]
        Left [node] Right [node]
    where
        Search [= Key]
    by
        Key Val [transform]
        Left Right
end function
```

Adding If clauses to TXL

```
function findApply SearchKey [number]
    replace [node]
        Key [number] Val [value]
        Left [node] Right [node]
    if where
        SearchKey [< Key]
    then by
        Key Val Left [findApply SearchKey] Right
    else if where
        SearchKey [> Key]
    then by
        Key Val Left Right [findApply SearchKey]
    else by
        Key Val [transform] Left Right
    end if
end function
```

Generics

```
rule sort T [type] LessThan [rule [T]]
    replace [repeat T]
        N1 [T] N2 [T] Rest [repeat T]
    where
        N2 [LessThan N1]
    by
        N2 N1 Rest
end rule

...
    construct Sorted [repeat pair]
        Pairs [sort [pair] pairLess]
```

Pattern Abstraction

```
rule assignPat : Id [id] Expr [expr]
    match [statement]
        assign( Id [id], Expr [expr] );
    where
        Id [needToRewrite]
    where
        Expr [isConst]
end rule

rule rewriteAssignment
    replace [statement]
        Statement [statement]
    where
        Statement [assignPat : Id [id] Expr [expr]]
    by
        Id = Expr;
end rule
```

Pattern Parameterization

```
function getAssign : Id [id] Expr [expr]
    match [statement]
        Id [id] = Expr [expr];
end function

rule genericReplace AssignPat [rule : [id] [expr]]
    replace [statement]
        Stmt [statement]
    where
        Stmt [AssignPat : Id [id] Expr [expr]]
    by
        Id [_ 'set] ( Expr );
end rule
...
    by
        Program [genericReplace getAssign]
```

Modularity

- TXL has no language facilities to support collaboration
- Programmers must invent their own ways of
  - Avoiding name collisions
  - Defining interfaces

Modularity

- Added a modularity feature to TXL
- Programmers can define modules, which encapsulate
  - Rules
  - Grammar definitions
  - Global variables

Modularity

```
module HTML
    public
        boldize
    end public

    define item
        [begin_tag] [any] [opt end_tag]
    end define

    define begin_tag
        < [id] [repeat option] >
    end define

    define end_tag
        < / [id] >
    end define

    function boldize
        replace [any]
            A [any]
        construct BoldTag [begin_tag]
            <B>
        by
            A [tagwith BoldTag]
    end function
end module

rule tagIds
    replace $ [id]
        Id [id]
    by
        Id [HTML.boldize]
end rule
```