Term Rewriting with Type-safe Traversal Functions

M.G.J. van den Brand
P. Klint
J.J. Vinju
Overview

• Transforming programs with rewriting
  – Ups and downs
• Traversal functions
  – Intro, examples
• Experience
• Functions and strategies
• Conclusions
• Questions
Transforming with rewriting

• Software maintenance, renovation
  – A lot of small changes
  – Extensive restructuring
  – Dialect/language translations

• Compiler construction
Transforming with rewriting

• Many sorted term rewriting is a practical tool:
  – Matching (modulo A or AC)
  – Construction
  – Normalization

• Swift, safe and simple
  – Compilers and fast interpreters available
  – Types
  – Nothing but rules
Transforming with rewriting

• Simple rewrite rules not powerful enough:
  – No context information
  – Confluence and termination
  – Sort preserving
• No context sensitive translations
• No compositionality
• No analysis
Example 1 – using functions

context-free syntax

Exp + Exp \rightarrow Exp
foo(Exp, Nat) \rightarrow Exp
trans(Exp) \rightarrow Nat

equations

[] A + B = B + A
[] foo(A + B, N) = B + A + N
[] trans(A + B) = 1 + trans(A) + trans(B)
Functions

• Extra functions solve many problems
  – Confluence and termination become easier
  – Compositional
  – Non type-preserving
  – Extra arguments for context information

• Some practical properties are kept:
  – Type-safe, matching, construction, just rules

• But: no normalization (term traversal)
Example 2 – Y2K in COBOL

- COBOL
  - AROUND 500 PRODUCTIONS
  - AROUND 120 SORTS

- Y2K transformation
  - Local
  - Widespread
Example 2 – Y2K in COBOL

imports COBOL

context-free syntax

  fix-y2k(COBOL,OPTIONS) -> COBOL

  ...

  fix-y2k(EXPR,OPTIONS)  -> EXPR

equations

  [] fix-y2k(DATA DIVISION ... PROCEDURE DIVISION ... ) =

    DATA DIVISION ... PROCEDURE DIVISION fix-y2k(...)

  [] ...

  [] fix-y2k(expr,opts)  = do-some-fix(expr,opts)
Example 3 – the essence

imports COBOL

context-free syntax

  fix-y2k(COBOL, OPTIONS)  ->  COBOL
  fix-y2k(EXPR, OPTIONS)    ->  EXPR

equations

  [] fix-y2k(expr, opts)    =  do-some-fix(expr, opts)
Traversal functions

• Functions that traverse a tree
• Attributes that define traversal behavior:
  – Top-down, bottom-up, …
• Attributes that define the type of traversal
  – Sort preserving, maps, …

current-free syntax

  Func(Tree, ...) -> Tree {traversal(attributes)}
  Func(Tree, Value, ...) -> Value {traversal(...)}
Example 4 – a Traversal Function

```plaintext
imports COBOL
context-free syntax

  fix-y2k(COBOL,OPTIONS)  ->  COBOL  {traversal(…)}
  fix-y2k(EXPR,OPTIONS)  ->  EXPR  {traversal(…)}

equations

  fix-y2k(expr,opts)  =  do-some-fix(expr,opts)
```

• The tedious parts have disappeared

• They are defined by the traversal attributes
Example 4 – a Traversal Function
Example 4 – a Traversal Function

fix-y2k(COBOL)

DD

PD

EXPR

EXPR

EXPR

EXPR
Example 4 – a Traversal Function
Traversal Orders

Top-down

Bottom-up

Continue

Break

Right-to-left

Left-to-right
Traversals Types

- **Sort preserving Transformers**
  
  \[-F(S,...) \rightarrow S \{\text{traversal(trafo,...)}\}\]

- **Accumulators** map a tree to a single sort
  
  \[-F(S,S',...) \rightarrow S' \{\text{traversal(accu,...)}\}\]

- The combination does both:
  
  \[-F(S,S',...) \rightarrow S#S' \{\text{traversal(accu,trafo,...)}\}\]
Traversals Types

• Trafo, accu are type-safe:
  – Trafo does not change the type of any tree
  – Accu does not change the tree, only the accumulated value of a fixed type

• Typed term representation at runtime
  – To be able to match specific node types
Experience

- COBOL tools
  - GOTO removal (extensive restructuring)
  - Dialect migrations, deprecated feature removal
  - Large systems (e.g. 440,000 LOC)

- SDF2 statical property checker
  - Checks properties of grammars like completeness etc.

- Java code smell detection
  - Detects and visualizes bad practices in Java code by transforming Java to Rigi standard format
Functions and Strategies

• Strategies:
  – Composed from first-class combinators
  – Are applied to rules
  – Make evaluation process explicit

• Functions
  – Are defined as rewrite rules
  – Use implicit evaluation mechanism (e.g. innermost)

• Same effect, different means
Traversal Functions and Strategies

• Traversal combinators
  – Difficult typing issues
  – More specific traversal scenarios expressible
  – No straightforward context information

• Traversal functions
  – No type system extension needed
  – General traversal scenarios expressible
  – Easily carry context information
Conclusions

• Term Rewriting with Traversal Functions
• A practical extension of term rewriting
• Simple
• Type-safe
• Efficient
Questions

http://www.cwi.nl/projects/MetaEnv