TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

Winter term 2015/2016 - Model-Driven Software Development in Technical Spaces

Exercise 4: Attribute Grammars with RACR

In this exercise, Reference Attribute Grammars (RAGs) and the tool RACR[1] for Reference Attribute Grammar controlled Rewriting are introduced. A simple grammar for mathematical expressions is defined and evaluated. RACR supports efficient rewriting of syntax trees of a grammar, which is used here to optimize the mathematical expressions.
RACR is a library for the programming language Scheme, but it also has bindings for C# and the .NET framework. The following task can be solved in either language. Due to technical limitations, .NET is suggested for development under Windows, but for the given example, it should also run with mono under Linux and OSX. The Scheme library is currently not supported under Windows.

Task 1

Develop a RAG specification for a small expression language. Use and extend the specification from the RACR.NET documentation[2]. An initial implementation that can be extended is available[3].

- The expression language should support some more operators: `+`, `*`, `/`, and both the unary and binary `-`.
- The eval attribute has to be extended for those operators.
- An attribute for pretty-printing the expressions has to be written.

Task 2

The second task is rewriting the syntax tree. This is a common use case in compiler construction, where constant expressions like `3 + 4` are rewritten to `7` during the compilation.

- Implement the folding of sub-trees that only contain numbers.

Another optimization that can be done is the removal of sub-trees that are a multiplication with zero, i.e. `((3 * X) * 0)`.

- Implement the the replacement of multiplication by zero.

Structure of the submission

The resulting program should be either a single Scheme or C# file. The program should do the following:

- Create a RACR specification containing the above-mentioned operators as well as constant declarations (please extend the RACR.NET example).
- Create a tree containing all operators and optimization cases.

[1] `https://github.com/christoff-buerger/racr`
[2] `https://github.com/christoff-buerger/racr/blob/master/racr-net/documentation/racr-net-manual.pdf`
[3] `https://github.com/2bt/racr/tree/master/profiling/mathexp`

- Print this tree.
- Run the first optimizer, then print again.
- Run the second optimizer, the print again.

If there are problems, please ask me `johannes.mey@tu-dresden.de`
The tutorial to JastAddi[4], another RAG system also provides a nice introduction to RAGs.

[4] `http://jastadd.org/web/documentation/tutorial.php`