TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology, Software Technology Group

WS2017/18 – Model-driven Software Development in Technical Spaces

Reference Attribute Grammars with RACR

Professor: Prof. Dr. Uwe Aßmann
Tutor: Dr.-Ing. Thomas Kühn

1 RACR

In this exercise, Reference Attribute Grammar Controlled Rewriting (RACR) [1, 2] is introduced. It is a library for the programming language Scheme, which is currently only supported in Linux and OSX. In this exercise, a simple grammar for mathematical expressions is defined and evaluated. RACR supports efficient rewriting of syntax trees of a grammar, which is used later to optimize mathematical expressions.

1.1 Task 1: Extension of Expression Language

Develop a RAG specification for a small expression language, that can evaluate simple expressions. An initial specification comprising grammar (for addition, multiplication, constant numbers and variables) and some attributes are given. The first task is to extend the specification to support unary and binary minus, binary division and pretty-printing. To achieve this, the following subtasks are needed:

- Install and understand RACR. Read the README.
- Extend the grammar enabling new nonterminal types. Use `DivExp`, `MinusExp` and `UnaryMinusExp` as names.
- Implement new attributes for the operations and pretty-printing.

Pretty-printing does not need to include the list of variable definitions, thus it only prints the expression. Furthermore, all binary and unary expressions must be enclosed in brackets.

Hint: Use `string-append` and `number->string` as needed. Printing the model Task1 should yield "`(((a+1.0)*(1.0/(-z)))-12.0)`".

1.2 Task 2: Rewrite to optimize

Given a generated tree, perform two optimization: constant folding and multiplicative annhilator detection.

In the first case, expressions comprising only constant values are rewritten to the constant value they (always) evaluate to, e.g., `(3 + 4)` is rewritten to `7`. The second

case will rewrite multiplications with zero, i.e., it will rewrite `((4 + a) * 0)` to `0`. To complete the second task, the following subtasks are needed:

- Implement pattern attributes matching foldable, constant expressions, and multiplications with zero.
- Implement rewrites for both cases by extending the method `optimize`.
- Test with a small example checking the AST before and after optimization (using the attribute `Print` from Task 1.1).

The only file to be submitted is the specification file `main.scm`.

1.3 Additional Information and Links

- The Scheme implementation Racket: `https://racket-lang.org/download/`
- RACR source code: `https://github.com/christoff-buerger/racr`
- RACR documentation – AST rules – Rewriting

References

[1] Christoff Bürger. RACR: A Scheme library for reference attribute grammar controlled rewriting. Technical Report TUD-Fl12-09, Technische Universität Dresden, Dresden, 2012. URL `http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-104623`.

[2] Görel Hedin. Reference attributed grammars. *Informatica (Slovenia)*, 24(3), 2000. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.8792&rep=rep1&type=pdf`.