A rewriting-strategies-based tool for transforming process-algebraic equations

Sergey V. Goncharov[1], Arseniy Y. Rudich[1], and Yaroslav S. Usenko[2]

[1] Department of Theoretical Cybernetics,
National Taras Shevchenko University of Kyiv, Ukraine
[2] Laboratory for Quality Software,
Technical University of Eindhoven, The Netherlands

1 Introduction

In this abstract we describe an attempt to implement a *linearization* algorithm [?] for micro Common Representation Language (μCRL) [?].

This algorithm allows transforming specifications in μCRL to "simpler" ones, containing just one Linear Process Equation (LPE). For the implementation we use the program-transformation system STRATEGO [?] that is based on *rewriting strategies*. STRATEGO allows to specify complex term-rewriting systems and their application strategies, and to generate efficient C code from them, which can be further compiled and executed. The implementation has been tested on a set of examples of μCRL specifications, and the results were compared with an existing implementation of μCRL linearization in C [?].

μCRL is an algebraic specification language based of ACP style process algebra [?]. A distinct feature of μCRL in comparison with many other process algebras is that it offers a uniform equational framework for specification of data and processes. Data are specified by equational specifications: one can declare sorts and functions on these sorts, and describe the meaning of these functions by equational axioms. Processes are described using data-parametric systems of equations that describe process behavior using process-algebraic operations extended with constructs for conditional composition and data-parametric choice.

Linear Process Equations (LPE) is an interesting subclass of systems of recursive equations, which contain only one linear equation of special form. As it turns out, the restriction to LPE format still yields an expressive setting. For instance, in the design and construction of verification tools for μCRL, LPEs establish a basic and convenient format that can be seen as a symbolic representation of Labeled Transition Systems (LTSs).

The algorithms for reducing μCRL specifications to the linear form are described in details in [?]. Each particular algorithm consists of a chain of transformation steps that yield an equivalent μCRL specification with a form coming closer to the desired linear one. Each transformation step is, in essence, an equivalent transformation of a system of process equations. The steps are formally described using term rewriting systems and different forms of their extensions to equation rewriting systems. In some steps extra equations or data types are added to the original μCRL specification.

For the implementation of the linearization algorithm we chose Stratego [?]. Strategois a system for the specification of fully automatic program transformation systems based on the rewriting strategies paradigm. Its language, aimed to operate with terms, is based on rewriting concepts [?]. It has clear and well thought-out syntax and semantics, and a very efficient implementation. Stratego is based on ATerm library [?] for term representation, which supports maximal sharing and efficient automatic garbage collection. Parsing of input programs to ATerms is done using the SDF toolset [?], which generates parsers for an extension of context-free grammars. All that makes writing clear, manageable and efficient programs transformation programs easy, and therefore motivates our choice of the implementation framework, in comparison with implementation in plain C.

2 Details of Implementation

The basic concept of Stratego is a notion of rewriting strategy that is a rule specifying how to apply given term rewriting system to a term. Every program in Stratego contains the main strategy that, in fact, transforms the input term into the output one. This main strategy is normally defined as a composition (nondeterministic choice, sequential composition, congruent closure and others) of simpler strategies.

For each linearization step described in [?], we generate a separate rewriting step (rewriting strategy), and then the main strategy is, in essence, a sequential composition of these steps.

3 Associative-Commutative Rewriting

In SDF grammar it is possible to specify that certain operations are commutative, associative, and/or idempotent; and the generated parser will use lists, bags, or sets to represent the groups of operations. For example, in μCRL, alternative composition (+) is an associative, commutative and idempotent operation, and sequential composition (.) is an associative operation (there are more of such operations). In the parse tree these operations will be represented by sets and lists, respectively. However, to perform term rewriting module AC, we need to adopt the rewrite rules to be able to deal with list and sets. As an example, if the following rewrite rules

$$\begin{aligned} \mathsf{a} + \mathsf{b} \cdot \mathsf{c} &\to \mathsf{a} \cdot \mathsf{c} + \mathsf{b} \cdot \mathsf{c} \\ \mathsf{a} + 0 &\to \mathsf{a} \\ 0 \cdot \mathsf{a} &\to 0 \end{aligned}$$

have to be applied modulo associativity and commutativity, then the corresponding list operations will have the following form:

$$\begin{aligned}
&\langle \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n, [\mathsf{b}_1, \mathsf{b}_2, \ldots, \mathsf{b}_k], \mathsf{a}_{n+2}, \ldots, \mathsf{a}_m \rangle \rightarrow \\
&\qquad \langle \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n, [\langle \mathsf{b}_1, \mathsf{a}_n, \ldots, \mathsf{a}_m \rangle, \langle \mathsf{b}_2, \mathsf{a}_n, \ldots, \mathsf{a}_m \rangle, \ldots, \langle \mathsf{b}_k, \mathsf{a}_n, \ldots, \mathsf{a}_m \rangle] \rangle \\
&[\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n, 0, \mathsf{a}_{n+2}, \ldots, \mathsf{a}_m] \rightarrow [\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_m] \\
&\langle \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n, 0, \mathsf{a}_{n+2}, \ldots, \mathsf{a}_m \rangle \rightarrow \langle \mathsf{a}_1, \mathsf{a}_2, \ldots \mathsf{a}_n \rangle
\end{aligned}$$

A procedure to obtain such lifting rules is rather straightforward. Whenever we have a binary associative operation in the left-hand-side of a rewrite rule, we consider its extension to n terms, not just 2. Such a transformation was sufficient for the case of our implementation; however it is not clear whether it can be applied in more general cases.

4 Efficient Implementation of Simple Rewriting

One of the steps in linearization procedure consists of an application of a set of simple term rewrite rules. It turns out that all terms in the left-hand-sides of these rules have *depth* not bigger than 2. Experiment shows that in this case the standard *innermost* rewriting strategy is suboptimal. To improve on this, a custom strategy, called *double-traversal*, is used instead.

To use this strategy we split the set of rewrite rules in two. The rules from one set are used when the rewriting process traverses the term *down*, and the other set is used when the process traverses *up*. Initially, the process traverses down from the top (as in the outermost strategy) and once a rule is applied, the process traverses one step up, and repeats itself recursively. The process stops, when it reaches all bottom (leaf) nodes and no rule can be applied any longer.

An interesting question is whether this strategy can be used with a wider class of the rewrite systems.

5 Experimental Results and Future Work

At this point most of the linearization steps from [**?**] have been implemented. Certain optimization steps, like "regular linearization" and "operations on LPEs", have still to be implemented. We tested our linearization procedure on the examples available in the μCRL Toolset distribution [**?**].

A good sign is that both the implementation in the μCRL Toolset distribution and the ours produce LPEs that lead to *bisimilar* transition systems. This was checked automatically using the bisimilarity checker from the μCRL Toolset. Another good sign is that the performance of the new implementation is approximately the same as of the existing one. However, we have still to try out more realistic examples to compare the performances of the two implementations.

The number of generated states and transitions from the LPEs produced by our implementation is sometimes substantially larger than in the original implementation. This has still to be investigated. It can be due to the fact that not all optimization steps have been implemented yet.

In the future we plan to implement all optimization steps that are described in the literature. It is also interesting to extend the implementation to cover the case of timed μCRL. Trying larger examples may lead to a need for optimization of the rewriting procedures.

Acknowledgments The initial idea of the implementation is due to Jaco van de Pol and Eelco Visser back in summer of 2001. Eelco has also helped us with STRATEGO. An earlier attempt to implement the linearization algorithm in STRATEGOwas done by Konstantin O. Savenkov in [**?**].