2nd International Workshop on Reduction Strategies in Rewriting and Programming

WRS 2002

(held in conjunction with RTA 2002)

Copenhagen, Denmark
July 21, 2002

Abstracts of papers accepted for presentation

The Sketch of a Polymorphic Symphony
Ralf Laemmel
CWI, Amsterdam

Functional strategies were previously defined as first-class generic functions which can traverse into terms while mixing uniform and type-specific behaviour. The first-class status is witnessed by a combinator style of generic programming. This symphony reconstructs functional strategies as an amalgamation of certain bits of parametric polymorphism, type case, polytypism, and overloading. We illustrate the expressivness and conciseness of this reconstruction by providing highly parameterized definitions of traversal schemes. The resulting style of generic programming is extremely lightweight and easy to use because it only involves two special combinators not yet present in standard functional programming. The reconstruction is geared towards Haskell, and it is supported by a generative tool YAGH---Yet Another Generic Haskell.

Regular Sets of Descendants by Leftmost Strategy
Pierre Réty and Julie Vuotto
LIFO, Orléans

For a constructor-based rewrite system R, a regular set of ground terms E, and assuming some additional restrictions, we build a finite tree automaton that recognizes the descendants of E, i.e. the terms issued from E by rewriting, according to leftmost strategy.

Strategies for Source-to-Source Constant Propagation
Karina Olmos and Eelco Visser
Univ. Utrecht

Data-flow optimizations are usually implemented on low-level intermediate representations. This is not appropriate for source-to- source optimizations, which reconstruct a source level program after transformation. In this paper we show how constant propagation, a well known data-flow optimization problem, can be implemented on abstract syntax trees in Stratego, a rewriting system extended with programmable rewriting strategies for the control over the application of rules and dynamic rewrite rules for the propagation of information.

An Abstract Böhm-normalization
John Glauert and Zurab Khasidashvili
UEA Norwich
Dept. of Computer Science, Bar-Ilan Univ.

In this paper, we study normalization by neededness with respect to 'infinite results', such as Böhm-trees, in an abstract framework of Stable Deterministic Residual Structures (SDRS). We formalize the concept of 'infinite results' as suitable sets of infinite reductions, and prove an abstract infinitary normalization theorem with respect to such sets. We also give a sufficient and necessary condition for existence of minimal normalizing reductions.

Pattern-driven Reduction in Haskell
William L. Harrison and Richard B. Kieburtz
Pacific Software Research Center, Oregon Health & Engineering Univ.
Dept. of Computer Science, Bar-Ilan Univ

Haskell is a functional programming language with nominally non-strict semantics, implying that evaluation of a Haskell expression proceeds by demand-driven reduction. However, Haskell also provides pattern matching on arguments of functions, in let expressions and in the match clauses of case expressions. Pattern matching requires data- driven reduction to the extent necessary to evaluate a pattern match or to bind variables introduced in a pattern. In this paper, we provide both an abstract semantics and a logical characterization of pattern-matching in Haskell and the reduction order that it entails.

Applying ELAN Strategies in Simulation Processors over Simple Architectures
Mauricio Ayala-Rincón, Rinaldi Maya Neto, Ricardo P. Jacobi, Carlos Llanos, Reiner Hartenstein
Univ. de Brasilia
IESB Brasilia
Univ. Kaiserslautern

Simulation of processors over simple architectures is an important technique for verifying previously to the

expensive hardware implementation techniques involved in new technologies. Arvind's group has illustrated how to describe processors by rewriting and introduced a technique for proving the correctness of elaborated processors with respect to basic ones. The correctness of a processor is proved by showing that its related rewriting system does all that other rewriting system, considered correct, does. Basic concepts of rewriting as noetherianity and confluence are relevant for showing adequability of these rewrite based processor descriptions. In Arvind group's approach, simulation of the described is not done directly over the rewriting systems but over standard hardware description languages like Verilog after translating these rewrite descriptions adequately. Her we show how rewriting logic may be applied for purely rewriting simulation of processors and other proposals as is the case of evaluating performance of important hardware aspects of processors. Environments like ELAN, that is the one we use, are sufficiently versatile to allow for adequate implementations and easy modifications that are intrinsically related with hardware properties like the size and control of reorder buffers and the method of predictions used by speculative processors.pattern-matching in Haskell and the reduction order that it entails.

Operational Semantics for Lazy Functional Logic Programs
Elvira Albert, Michael Hanus, Frank Huch, Javier Oliver, and Germán Vidal
DSIC, Univ. Politécnica de Valencia
CAU Kiel

In this paper we define an operational semantics for lazy functional logic programs including notions like sharing, concurrency, non- determinism, etc. Such a semantic description is not only important to provide appropriate language definitions to reason about programs and check the correctness of implementations but it is also a basis to develop language-specific tools, like program tracers, profilers, optimizers, etc. First, we define a ``big-step" semantics in natural style to relate expressions and their evaluated results. Since this semantics is not sufficient to cover concurrency, search strategies, or to reason about costs associated to particular computations, we also define a ``small-step" operational semantics covering the features of modern functional logic languages. Finally, we provide the correctness of the small-step operational semantics.

Term Rewriting with Type-safe Traversal Functions
Mark G.J. van der Brand, Paul Klint, and Jürgen J. Vinju
CWI, Amsterdam

Term rewriting is an appealing technique for performing program analysis and program transformation. Tree (term) traversal is frequently used but is not supported by standard term rewriting. In this paper, many-sorted first-order term rewriting is extended with automatic tree traversal by adding two primitive tree traversal strategies and complementing them with three types of traversals. These so-called traversal functions can be either top-down or bottom-up. They can be sort preserving, mapping to a single sort, or a combination of these two. Traversal functiona have a simple design, their application is type-safe in a first-order many-sorted setting and can be implemented efficiently. We describe the operational semantics of traversal functions and discuss applications.

WRS 2002