Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Symbolic Computation 40 (2005) 795–829
Journal of Symbolic Computation
www.elsevier.com/locate/jsc

Operational semantics for declarative multi-paradigm languages☆

Elvira Albert[a], Michael Hanus[b,*], Frank Huch[b], Javier Oliver[c], Germán Vidal[c]

[a] *DSIP, Complutense University of Madrid, E-28040 Madrid, Spain*
[b] *Institut für Informatik, CAU Kiel, D-24098 Kiel, Germany*
[c] *DSIC, Technical University of Valencia, E-46022 Valencia, Spain*

Received 14 January 2003; accepted 14 December 2004
Available online 2 March 2005

Abstract

Declarative multi-paradigm languages combine the most important features of functional, logic and concurrent programming. The computational model of such integrated languages is usually based on a combination of two different operational principles: narrowing and residuation. This work is motivated by the fact that a precise definition of an operational semantics including all aspects of modern multi-paradigm languages like laziness, sharing, non-determinism, equational constraints, external functions and concurrency does not exist. Therefore, in this article, we present the first rigorous operational description covering all the aforementioned features in a precise and understandable manner. We develop our operational semantics in several steps. First, we define a

☆ A preliminary version of this article appeared in the Proceedings of WFLP'02 [Albert, E., Hanus, M., Huch, F., Oliver, J., Vidal, G., 2002b. Operational semantics for functional logic languages. In: Proc. of the Int'l Workshop on Functional and (Constraint) Logic Programming. WFLP 2002. In: Electronic Notes in Theoretical Computer Science, vol. 76. Elsevier Science Publishers.] and WRS'02 [Albert, E., Hanus, M., Huch, F., Oliver, J., Vidal, G., 2002a. An operational semantics for declarative multi-paradigm languages. In: Proc. of the Int'l Workshop on Reduction Strategies in Rewriting and Programming. WRS 2002. In: Electronic Notes in Theoretical Computer Science, vol. 70(6). Elsevier Science Publishers.]. This work was partially supported by CICYT TIC 2001-2705-C03-01, by the Generalitat Valenciana under grants CTIDIA/2002/205 and GRUPOS03/025, by the ICT for EU–India Cross-Cultural Dissemination Project ALA/95/23/2003/077-054, by the MCYT under grants HA2001-0059 and HU2003-0003, and by the DFG under grant Ha 2457/1-2.

* Corresponding author. Tel.: +49 431 880 7271; fax: +49 431 880 7613.
E-mail addresses: elvira@sip.ucm.es (E. Albert), mh@informatik.uni-kiel.de (M. Hanus), fhu@informatik.uni-kiel.de (F. Huch), fjoliver@dsic.upv.es (J. Oliver), gvidal@dsic.upv.es (G. Vidal).

0747-7171/$ - see front matter
doi:10.1016/j.jsc.2004.01.001

natural (big-step) semantics covering laziness, sharing and non-determinism. We also present an equivalent small-step semantics which additionally includes a number of practical features like equational constraints and external functions. Then, we introduce a deterministic version of the small-step semantics which makes the search strategy explicit; this is essential for profiling, tracing, debugging etc. Finally, the deterministic semantics is extended in order to cover the concurrent facilities of modern declarative multi-paradigm languages. The semantics developed provides an appropriate foundation for modeling actual declarative multi-paradigm languages like Curry. The complete operational semantics has been implemented and used with various programming tools.

Keywords: Functional logic programming; Operational semantics

1. Introduction

Declarative multi-paradigm languages combine the most important features of functional programming (nested expressions, higher-order functions, efficient demand-driven computations, polymorphism), logic programming (logical variables, partial data structures, built-in search) and concurrent programming (concurrent computations with inter-thread synchronization and communication on logical variables). The computational model of such integrated languages is usually based on a seamless combination of two different operational principles: narrowing and residuation (see Hanus (1994) for a survey). *Narrowing* (Slagle, 1974) allows the instantiation of variables in expressions and then applies reduction steps to the function calls of the instantiated expressions. This instantiation is usually computed by unifying a subterm of the expression with the left-hand side of some program rule. On the other hand, *residuation* (Aït-Kaci et al., 1987) is based on the idea of delaying function calls until they are ready for a deterministic evaluation. Residuation preserves the deterministic nature of functions and naturally supports concurrent computations by employing dynamic scheduling.

This work is motivated by the fact that there is no existing precise definition of an operational semantics covering all aspects of modern declarative multi-paradigm languages. For instance, the report on the multi-paradigm language Curry (Hanus, 2003) contains a fairly precise operational semantics but covers sharing only informally. The operational semantics of the functional logic language Toy (López-Fraguas and Sánchez-Hernández, 1999) is based on narrowing and sharing but the formal definition is based on a narrowing calculus (González-Moreno et al., 1999) which does not include a particular pattern-matching strategy. However, the latter becomes important, e.g., if one wants to reason about costs of computations (see Antoy (2001) for a discussion about narrowing strategies and calculi). Defining a precise operational semantics for these languages is not an easy task since one must cover many different notions like sharing, logical variables, search strategies and concurrency, as well as the interactions among them.

Defining a rigorous operational semantics covering all aspects of actual multi-paradigm languages is a difficult but important task, not only for reasoning about programs and correctness of implementations but also for the development of implementation-oriented analyses and tools (like profilers, tracers, debuggers, partial evaluators). Well-known