From a Formal Dynamic Semantics of Sisal to a Sisal Environment

Extended Abstract[*]

I. Attali
INRIA BP 93
06902 Sophia Antipolis
France
ia@sophia.inria.fr

D. Caromel
I3S Univ. of Nice BP 145
06560 Sophia Antipolis
France
caromel@unice.fr

A. L. Wendelborn
Dept. Computer Science
University of Adelaide 5005
Australia
andrew@cs.adelaide.edu.au

1 Introduction

We present a formal definition of the dynamic semantics of a significant part of the language Sisal 2.0 in the structural operational style of Natural Semantics [6], namely Typol inference rules [3], within the Centaur system [2], a generic specification environment.

Sisal (Streams and Iteration in a Single Assignment Language) is a strongly typed, applicative, single assignment language in use on a variety of parallel processors, including conventional multiprocessors, vector machines and data-flow machines. Sisal 1.2 has been in use since 1984; Sisal 2.0, a new language definition, is currently under development [1]. Sisal includes higher-order functions, a limited form of polymorphism, overloading and type inference, powerful array operations, and streams for pipelined parallelism.

We expect, with a formal semantic description of Sisal, to provide a firm foundation for understanding and evaluating the language design issues, and provide a valuable reference for both implementors and programmers. Our ultimate goal is to formally describe (prove and extend) parallelization techniques and algorithms for Sisal compilation, and to incorporate such techniques into a program development and visualization environment for Sisal programming.

2 Centaur and the Natural Semantics

The Centaur system is a formal tool to model and implement all aspects of programming languages. From the specifications of the syntax and the semantics of a given language, one can automatically produce a syntactic editor and interactive semantic tools (type checkers, interpreters) for this language.

The general idea of a semantic definition in Natural Semantics is to provide axioms and inference rules that characterize semantic behaviors to be defined on language constructs (abstract syntax operators). Semantic definitions are mathematically precise and executable.

Natural Semantics descriptions have been extensively used to express dynamic semantics of various languages [7], define various translations, describe compiler optimizations and formalize the semantics of data flow graphs [8]. We have drawn on a natural semantic characterization of Lassi [4], a preliminary investigation into the semantics of a small Sisal-like language. Aspects of the development of a Sisal 2.0 compiler are described in [5].

3 Structure of the Semantics

The dynamic semantic specification is written upon an abstract syntax for the Sisal language which contains all Sisal constructs, semantic values and environments. Language constructs include function definitions, expressions such as binary operations, references to compound objects, as well as the *let* construct, selection and loop expressions.

Semantic values in Sisal are either constants, compound objects, or closures which are pairs of λ-expression (representing the function body) and environment. Environments are necessary to manage the binding between identifiers and semantic values since the value of an expression depends on the values of identifiers that occur free in it, including bindings for function names, coming from *let* or a *for* declarations.

[*] A full version of this paper is available as I3S research report number RR 94-49.

4 Semantic Definition

With the Typol formalism, it is possible to structure the semantic definition in separate modules, each of which deals with similar concerns. Therefore, we have a variety of Typol modules specifying the dynamic semantics of all constructs of the Sisal language, including values and environments.

The starting point of the semantics is to evaluate the main function body given its name and its arguments. This set merely relies on two other sets in order to find the function definition in the current program and to execute it.

The core of the language (expressions) is treated in a separate module which contains sets (for a list of expressions, identifiers, constants, evaluation of operations on scalar types) and calls to other specific modules for evaluation of function invocation, selection, array and stream operations, and *for* or *let* constructs. Primitives for manipulation of the environment are gathered in a specific module.

5 Semantics of streams

Streams in Sisal 2.0 are intented to be evaluated non-strictly in order to permit potentially infinite streams, pipelined parallelism, and input/output operations. In order to cope with these features, while our semantics is globally *strict* (call-by-value semantics), we define a non-strict semantics for streams. The streams remain fully unevaluated, as well as the expressions which use the stream values. One source of evaluation is a reduction operation applied to a stream; in this case, the stream is fully evaluated. A second source occurs at the top-level when one result of the main function is of type stream: the final result is computed in a demand-driven way; this allows production of the stream to be regulated by its consumer.

6 An Interactive Environment for Sisal

The Sisal programming environment includes a structure editor and an interpreter. The syntactic editor provides a guided editing mode as well as an inline textual editing mode. Abstract syntax is used to check the validity of editing operations.

When triggered, the interpretation of a Sisal program unfolds in an animated way and some visual help is provided; the current expression to be evaluated is highlighted and the user is always aware of the execution scheme of the program. The final result (multiple values) is displayed in a separate window.

7 Conclusions and Future Work

First, we intend to complete the dynamic semantics definition. We also want to build more tools which will help Sisal programmers in designing and understanding their programs.

Future work will include parallelizing tools for Sisal: we aim at specifying translation from Sisal into graph-oriented intermediate formats, and upon this format, interactive transformations for a parallel implementation. Finally, thanks to our formal framework, we aim at proving the validity of these transformations.

Acknowledgements

We are pleased to thank Jean-Luc Gaudiot and Yung-Syau Chen of the University of Southern California for their help in the specification of arrays and loops.

References

[1] Böhm A. P. W. *et al.* "Sisal Reference Manual (language version 2.0)" *Draft Report*, 1992.

[2] Borras P. *et al.* "CENTAUR: the system" *SIGSOFT'88, 3rd Annual Symposium on Software Development Environments*, Boston, 1988.

[3] Despeyroux T. "Typol: a formalism to implement Natural Semantics" *INRIA Technical Report 94*, 1988.

[4] Errington L. "Lassi Semantics", *Sisal project internal report, Dpt of Computer Science, Univ. of Adelaide*, 1991.

[5] Fitzgerald, S. M. "Copy Elimination for True Multidimensional Arrays in SISAL 2.0", in *Proc. of the Third SISAL Users and Developers Conference*, San Diego, 1993.

[6] Kahn G. "Natural Semantics" in *Proc. of Symp on Theoretical Aspects of Computer Science*, Passau, Germany, LNCS 247, 1987.

[7] Milner R., Tofte M. & Harper R. "The Definition of Standard ML", *MIT Press*, 1990.

[8] Pingali K. *et al.* "Dependence Flow Graphs: An Algebraic Approach to Program Dependencies" in *Proc. of POPL'91*, Orlando 1991.