The ABACO System
An Algebraic Based Action COmpiler

Hermano Perrelli de Moura*
Luis Carlos de Sousa Menezes**

Federal University of Pernambuco - Department of Informatics

Abstract. We show the main features of the ABACO system, a semantic directed compiler generator, that produces object-oriented compilers from action semantics descriptions of programming languages.

1 Introduction

Action semantics [2] is a useful formalism to describe the semantics of programming languages. Its modular structure and formal notation, based on terms of the English language, eases the writing of real programming languages' descriptions. Some examples of real programming languages described in action semantics are: PASCAL [3] and STANDARD ML [5].

Action semantics formal notation is based on an algebraic specification model known as unified algebras [1]. This model defines entities called sorts and operations that manipulate them.

An action semantic description for a programming language is divided in the following modules:

- Abstract Syntax: describes the abstract syntax for the programming language.
- Semantic Functions: describes a mapping from the abstract syntax tree (AST) of a program to its meaning. The meaning of a program uses action notation, the formal notation used in action semantics, to describe the program's semantics.
- Semantic Entities: defines the data types used by the language, and auxiliary sorts and operators used in the description.

2 The ABACO System

We built an automatic compiler generation tool named ABACO (Algebraic Based Action COmpiler). The system combines object-orientation and action semantics to produce implementations of programming languages from their action semantic descriptions. The system is composed by the following tools:

* E-mail:hermano@di.ufpe.br
** E-mail:lcsm@di.ufpe.br

– A generic *unified algebras translator*: this tool accepts unified algebras specifications and produces an object-oriented library that implements the given specification. The choice of object-oriented languages as target language is based on the similarities between the existing concepts in both models, for example the concepts of sort inclusion (in unified algebras) and class inheritance (in object-orientation).
– A *parser generator*: this tool accepts an action semantic description of a programming language and builds a description that can be used with parsers generators like Lex and Yacc to produce the parser for the specified language.
– An *action compiler*: generates a C++ code which implements an action that represents the meaning of a program in the described language (program action).

The process of building a compiler for a described programming language has the following steps, showed graphicaly in Figure 1:

1. The definition of the abstract syntax of the programming language is processed by the *parser generator* that will generate the parser. The generated parser recognizes the source programs and produces their abstract syntax trees (AST).
2. The programming language description is processed by the *unified algebras translator* that will produce a library, named dynamic library, that will be able to give the meaning of a program, represented as an AST and produced by the parser, according the action semantics description language's.
3. The programs writen in the specified language will need a library that defines the data types used by the language to be correctly compiled. This library, named static library, is obtained processing the semantics entities of the programming language description with the unified algebras translator.
4. The dynamic library is linked with the *action compiler* to produce the code generator for the specified language. This program is able to produce C++ programs from AST produced by the parser generated in step 1. The C++ programs generated by the code generator can be compiled using a generic C++ compiler.

The generated compiler is formed by the parser, the code generator, the static library and a generic C++ compiler. Its architeture and compilation process is showed in Figure 2.

3 Conclusions

The Abaco system is useful to prototype semantic descriptions of programming languages. His main characteristics is to produce a dynamic implementation for the datatypes of the described language. It enables the generated compiler to represent more accurately the peculiarities of the specified programming language.

A prototipe of this system was implemented using the C++ language and was tested using the GNU C++ compiler (GCC) in the Solaris operating system but

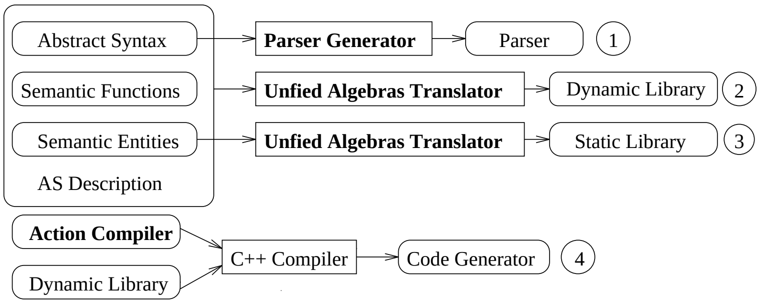

Fig. 1. Compiler Generation Using the ABACO System

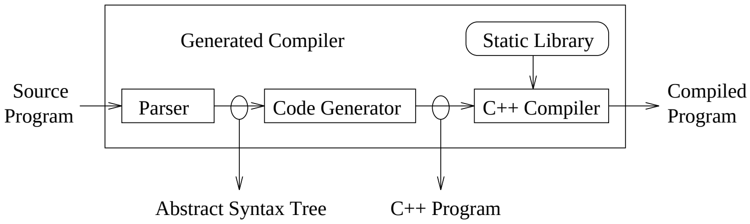

Fig. 2. Architeture of the Generated Compiler

it could work in every system supporting the C++ language. The source code of this implementation can be obtained at the Recife Action Tools home page [4].

References

1. P. D. Mosses. Unified algebras and modules. Departamental Report DAIMI PB–266, Aarhus University, Computer Science Department, Denmark, 1988.
2. Peter D. Mosses. *Action Semantics*. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.
3. Peter D. Mosses and David A. Watt. Pascal action semantics, version 0.6. Available by FTP as ftp.brics.dk/pub/BRICS/Projects/AS/Papers/MossesWatt93DRAFT/-pas-0.6.ps.Z, March 1993.
4. Homepage of the RAT Project. http://www.di.ufpe.br/∼rat.
5. David A. Watt. An action semantics of Standard ML. In *Proc. Third Workshop on Math. Foundations of Programming Language Semantics, New Orleans*, volume 298 of *Lecture Notes in Computer Science*, pages 572–598. Springer-Verlag, 1988.