Object Oriented Semantics Directed Compiler Generation: A Prototype

Luiz Carlos Castro Guedes[1] Edward Hermann Haeusler[2] José Lucas Rangel[2]

The tool presented here is a prototype of a compiler generator centered on the denotational description of the source language. It is based on an object oriented translation model. This model maps the denotational definition of a programming language into an Object Oriented Programming Language with separated hierarchies for specifications (types) and implementations (classes). The central idea in the model is the mapping from syntactic and semantic domains into types, and from semantic functions into methods of those types. Semantic equations are mapped to the implementation of some of those methods. Its correctness has already been proved and a system prototype implemented. The entire translation process may be summarized by the following table:

Description Components	Examples	Resulting Code
Semantic Domains	δ:D	type tD; class cD;
Syntactic Domains	Δ:Dom	type Dom;
Semantic Functions	$\mathcal{F}$: Dom$\rightarrow$ $D_1 \rightarrow \ldots \rightarrow D_s \rightarrow$ D	type Dom is tD $\mathcal{F}$ (tD_1 δ_1, ..., tD_s δ_s); end;
Semantic Equations	$\mathcal{F}$[[Rule]](δ_1, δ_2, ..., δ_s) = B	tD $\mathcal{F}$ (tD_1 δ_1, ..., tD_s δ_s) B_n end;
Syntactic Rules	Rule: $\Delta = \Delta_1\ \Delta_2 \ldots \Delta_m$	class Rule of Dom variables : Dom_1 Δ_1; ... Dom_m Δ_m; methods: tD $\mathcal{F}$ (tD_1 δ_1, ..., tD_s δ_s) B_n end; end;

Table 1 - Translation Process Summary

The compiler generator has two major components: the Tree-Building-Parser Generator (TBPG) and the Denotational Compiler Generator (DCG), see figure 1.

The TBPG was based upon the syntactic analysis ascent method R*S [1], which ignores single production rules in a efficient way, producing smaller derivation trees and, thus, smaller abstract syntax trees (AST).

The DCG reads the standard semantics of the desired compiler input language, *L*, and produces four files with the declaration of classes and definition of methods produced for either the Syntactic Domains and Semantic ones. Declarations of classes are at the C-style header files *L*.HSY and *L*.HSE and definition of their methods at the files *L*.CSY and *L*.CSE, respectively, where the suffix SY stands for syntactic constructions and SE for semantic ones.

Thus, a compiler (generated for a language L) works in three passes:

1. reading a program written in a language *L* and building its AST representation.
2. traversing the AST and producing its denotational representation.
3. compiling the denotational representation and linking it with semantic classes, producing an executable representation of the input program

It is interesting to observe that semantic classes are involved not only at run time, but also at compile time, since the methods of the syntactical classes (classes that correspond to syntactical domains) must know which classes they are generating code for.

[1]Universidade Federal Fluminense
[2]Pontifícia Universidade Católica do Rio de Janeiro
e-mail:{guedes, hermann, rangel}@inf.puc-rio.br
addr: Pontifícia Universidade Católica do Rio de Janeiro, Depto. de Informática, R. Marquês de S. Vicente 225, RDC, 4º andar, Gávea, Rio de Janeiro, RJ, BRAZIL, CEP 22453-900

The DCG prototype (written itself in C++) generates C++ code for classes which represent the denotations of the parts of the source program. Although it does not support separate hierarchies for types and classes, specifications can be written as abstract super-classes. A class that implements a type turns into a sub-class of the abstract class corresponding to the type. C++ seemed the best choice because it is the most powerful language available nowadays capable of simulating separate hierarchies and those functional features required by translation. Its high portability is another outstanding point.

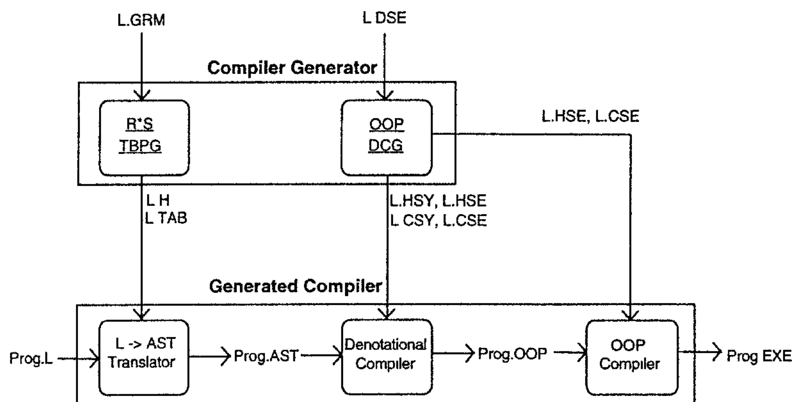

Figure 1 - Compiler Generation Process

Performance tests have shown our system outperforms similar systems [2,3,4,5] and that its produced compilers are just one order of magnitude slower than hand written ones. The great efficiency of the model is a promising step towards the automatic generation of production quality compilers. Thus, a natural and elegant model to translate programming language descriptions into realistic compilers with its correctness guaranteed has been obtained.

	with range checking		without range checking	
	Bubble Sort	Prime Number	Bubble Sort	Prime Number
Turbo Pascal V5.5	7.65 s	6.65 s	1.65 s	3.65 s
Produced Compiler	37 s	49 s	29 s	49 s
Slow-down	4.84	7.37	17.57	13.42

Table 2 - Execution time on the Cx80486 DLC

Table 1 shows execution times for a bubble sort algorithm on a 1000 integers array and for finding the 1000th prime number on a Cx80486 DLC computer with 40MHz of clock.

References

1. Schneider, S.M., "Gramáticas e Reconhecedores R*S(k)", D.Sc. Thesis, Universidade Federal do Rio de Janeiro, Brazil, 1987.
2. Mosses, P.D., "SIS - Semantic Implementation System", Technical Report Daimi MD-30, Computer Science Department, Aarhus University, 1979.
3. Moura, H.; Watt, D.; "Action Transformations in the ACTRESS Compiler Generator"; Compiler Construction - 5th International Conference CC94; Lecture Notes in Computer Science; vol. 786; Springer-Verlag; pp:16-30; 1994
4. Orbaek, P.; "OASIS: An Optimising Action-Based Compiler Generator"; Proceed. of the First International Workshop on ACTION SEMANTICS; Edinburgh, Scotland; 1994; BRICS Notes Series NS-94-1; pp: 99-114.
5. Palsberg, J., "Provably Correct Compiler Generation", Ph.D. Thesis, Aarhus University, Denmark, 1992.