Compiler Construction using LOTOS NT

Hubert Garavel, Frédéric Lang, Radu Mateescu

INRIA Rhône-Alpes - VASY
655, avenue de l'Europe - F-38330 Montbonnot, France
{Hubert.Garavel,Frederic.Lang,Radu.Mateescu}@inria.fr

1 Introduction

Much academic and industrial effort has been invested in compiler construction. Numerous tools and environments[1] have been developed to improve compiler quality while reducing implementation and maintenance costs.

In the domain of computer-aided verification, most tools involve compilation and/or translation steps. This is the case with the tools developed by the VASY team of INRIA Rhône-Alpes, for instance the CADP[2] [5] tools for analysis of protocols and distributed systems. As regards the lexical and syntax analysis, all CADP tools are built using SYNTAX [3], a compiler generator that offers advanced error recovery features. As regards the description, construction, and traversal of abstract syntax trees (ASTS), three approaches have been used successively:

– In the CAESAR [8] compiler for LOTOS [10], ASTS are programmed in C. This low-level approach leads to slow development as one has to deal explicitly with pointers and space management to encode and explore ASTS.
– In the CAESAR.ADT [6] and XTL [13] compilers, ASTS are described and handled using LOTOS abstract data types, which are then translated into C using the CAESAR.ADT compiler itself (bootstrap); yet, for convenience and efficiency, certain imperative processings are directly programmed in C. This approach reduces the drawbacks of using C exclusively, but suffers from limitations inherent to the algebraic specification style (lack of local variables, of sequential composition, etc.).
– For the TRAIAN and SVL 1.0 compilers, and for the EVALUATOR 3.0 [14] model-checker, the FNC-2[3] [12] compiler generator based on attribute grammars was used. FNC-2 allows to declare attribute calculations for each AST node and evaluates the attributes automatically, according to their dependencies. Although we have been able to suggest many improvements incorporated to FNC-2, it turned out that, for input languages with large grammars, FNC-2 has practical limitations: development and debugging are complex, and the generated compilers have large object files and exhibit average performances (slow compilation, large memory footprint due to the creation of multiple ASTS and the absence of garbage collection). Therefore, the VASY

[1] An extensive catalog can be found at http://catalog.compilertools.net
[2] http://www.inrialpes.fr/vasy/cadp
[3] http://www.inrialpes.fr/vasy/fnc2

team switched to a new technology in order to develop its most recent verification tools.

2 Using LOTOS NT for compiler construction

E-Lotos (*Enhanced* Lotos) [11] is a new Iso standard for the specification of protocols and distributed systems. Lotos NT [9, 16] is a simplified variant of E-Lotos targeting at efficient implementation. It combines the strong theoretical foundations of process algebras with language features suitable for a wide industrial use. The data part of Lotos NT significantly improves over the previous Lotos standard [10]: equational programming is replaced with a language similar to first-order Ml extended with imperative features (assignments, loops, etc.).

A compiler for Lotos NT, named Traian,[4] translates the data part of Lotos NT specifications into C. Used in conjunction with a parser generator such as Lex/Yacc or Syntax, Traian is suitable to compiler construction:

– Lotos NT allows a straightforward description of Asts: each non-terminal symbol of the grammar is encoded by a data type having a constructor for each grammar rule associated to the symbol. Traversals of Asts for computing attributes are defined by recursive functions using "case" statements and pattern-matching.
– Traian generates automatically "printer" functions for each Lotos NT data type, which enables to inspect Asts and facilitates the debugging of semantic passes.
– Traian also allows to include in a Lotos NT specification external data types and functions implemented in C, enabling an easy interfacing of Lotos NT specifications with hand-written C modules as well as C code generated by Lex/Yacc or Syntax.

3 Applications

Since 1999, Lotos NT has been used to develop three significant compilers. For each compiler, the lexer and parser are built using Syntax and the Asts using Lotos NT. Type-checking, program transformation, and code generation are also implemented in Lotos NT. Some hand-written C code is added either for routine tasks (e.g., parsing options) or for some specialized algorithms (e.g., model-checking):

– The Svl 2.0 [7] compiler transforms high-level verification scripts into Bourne shell scripts (see Figure 1).
– The Evaluator 4.0 model-checker transforms a temporal logic formula into a boolean equation system solver written in C; the solver is then compiled and executed, taking as input a labelled transition system and producing a diagnostic (see Figure 2).

[4] `http://www.inrialpes.fr/vasy/traian`

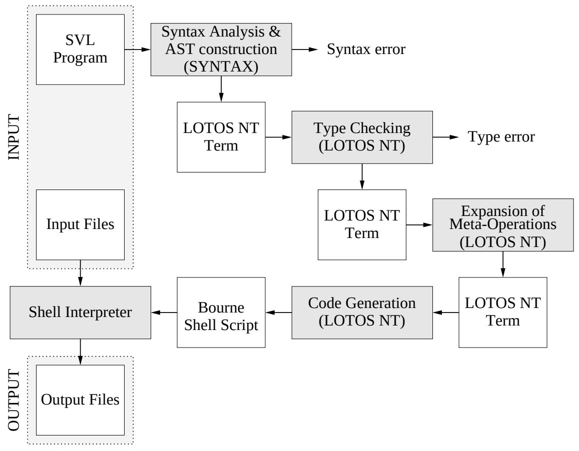

Fig. 1. Architecture of the SVL 2.0 compiler

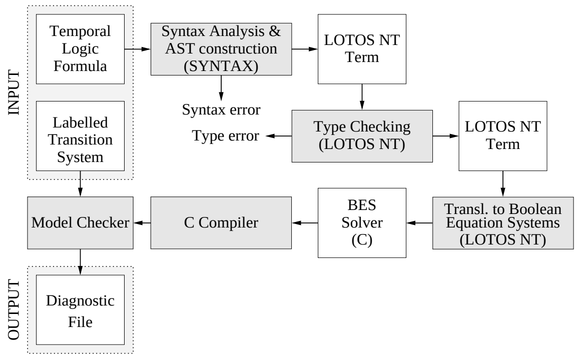

Fig. 2. Architecture of the EVALUATOR 4.0 model-checker

– The NTIF tool suite deals with a high-level language for symbolic transition systems; it includes a front-end, the NT2IF back-end generating a lower-level format, and the NT2DOT back-end producing a graph format visualizable by AT&T's GRAPHVIZ package.

The table below summarizes the size (in lines of code) of each compiler.

	SYNTAX	LOTOS NT	C	Shell	Total	Generated C
SVL 2.0	1,250	2,940	370	2,170	6,730	12,400
EVALUATOR 4.0	3,600	7,500	3,900	—	15,000	37,000
NTIF	1,620	3,620	1,200	—	6,440	20,644

4 Related work and conclusions

Alternative approaches exist based upon declarative representations, such as attributed grammars (Fnc-2 [12], SmartTools [1]), logic programming (Ale [4], Centaur [2]), or term rewriting (Txl[5], Kimwitu [18], Asf+Sdf [17]). In these approaches, Asts are implicit (not directly visible to the programmer) and it is not necessary to specify the order of attribute evaluation, which is inferred from the dependencies. On the contrary, our approach requires the explicit Ast specification and attribute computation ordering. Practically, this is not too restrictive, since the user is usually aware of these details.

Lotos NT is an hybrid between imperative and functional languages. Unlike the object-oriented approach (e.g., JavaCC[6]), in which Asts are defined using classes, and visitors are implemented using methods, the Lotos NT code for computing a given attribute does not need to be split into several classes, but can be clearly centralized in a single function containing a "case" statement. Compared to lower-level imperative languages such as C, Lotos NT avoids tedious and error-prone explicit pointer manipulation. Compared to functional languages such as Haskell or Caml[7] (for which the Happy[8] and CamlYacc parser generators are available), Lotos NT does not allow higher-order functions nor polymorphism. In practice, we believe that these missing features are not essential for compiler construction; instead, Lotos NT provides useful mechanisms such as strong typing, function overloading, pattern-matching, and sequential composition. Lotos NT external C types and functions make input/output operations simpler than Haskell/Happy, in which one must be acquainted with the notion of *monads*. Contrary to functional languages specifically dedicated to compiler construction such as Puma[9] and Gentle [15], Lotos NT is a general-purpose language, applicable to a wider range of problems.

The Lotos NT technology can be compared with other hybrid approaches such as the App[10] and Memphis[11] preprocessors, which extend C/C++ with abstract data types and pattern-matching. Yet, these preprocessors lack the static analysis checks supported by Lotos NT and Traian (strong typing, detection of uninitialized variables, exhaustiveness of "case" statements, etc.), which significantly facilitate the programming activity.

Our experience in using Lotos NT for developing three compilers demonstrated the efficiency and robustness of this pragmatic approach. Since 1998, the Traian compiler is available on several platforms (Windows, Linux, Solaris) and can be downloaded on the Internet. The three Traian-based compilers are or will be available soon: Svl 2.0 is distributed within Cadp 2001 "Ottawa"; Evaluator 4.0 and Ntif will be released in future versions of Cadp. Ntif

[5] `http://www.thetxlcompany.com`
[6] `http://www.webgain.com/products/java_cc`
[7] `http://caml.inria.fr`
[8] `http://www.haskell.org/happy`
[9] Puma belongs to the Cocktail toolbox (`http://www.first.gmd.de/cocktail`)
[10] `http://www.primenet.com/~georgen/app.html`
[11] `http://memphis.compilertools.net`

is already used in a test generation platform for smart cards in an industrial project with SCHLUMBERGER.

References

1. I. Attali, C. Courbis, P. Degenne, A. Fau, D. Parigot, and C. Pasquier. SmartTools: A Generator of Interactive Environments Tools. In *Proc. of CC '2001*, volume 2027 of *LNCS*, 2001.
2. P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. Centaur: the system. In *Proc. of SIGSOFT'88, 3rd Symposium on Software Development Environments (SDE3)*, 1988.
3. P. Boullier and P. Deschamp. Le système SYNTAX : Manuel d'utilisation et de mise en œuvre sous Unix. `http://www-rocq.inria.fr/oscar/www/syntax`, 1997.
4. B. Carpenter. The Logic of Typed Feature Structures. *Cambridge Tracts in Theoretical Computer Science*, 32, 1992.
5. J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In *Proc. of CAV '96*, volume 1102 of *LNCS*, 1996.
6. H. Garavel. Compilation of LOTOS Abstract Data Types. In *Proc. of FORTE'89*. North-Holland, 1989.
7. H. Garavel and F. Lang. SVL: A Scripting Language for Compositional Verification. In *Proc. of FORTE'2001*. Kluwer, 2001. INRIA Research Report RR-4223.
8. H. Garavel and J. Sifakis. Compilation and Verification of LOTOS Specifications. In *Proc. of PSTV'90*. North-Holland, 1990.
9. H. Garavel and M. Sighireanu. Towards a Second Generation of Formal Description Techniques – Rationale for the Design of E-LOTOS. In *Proc. of FMICS'98*, Amsterdam, 1998. CWI. Invited lecture.
10. ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, 1988.
11. ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, 2001.
12. M. Jourdan, D. Parigot, C. Julié, O. Durin, and C. Le Bellec. Design, Implementation and Evaluation of the FNC-2 Attribute Grammar System. *ACM SIGPLAN Notices*, 25(6), 1990.
13. R. Mateescu and H. Garavel. XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. In *Proc. of STTT '98*. BRICS, 1998.
14. R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. In *Proc. of FMICS'2000*, 2000. INRIA Research Report RR-3899. To appear in Science of Computer Programming.
15. F. W. Schröer. *The GENTLE Compiler Construction System*. R. Oldenbourg Verlag, 1997.
16. M. Sighireanu. LOTOS NT User's Manual (Version 2.1). INRIA projet VASY. `ftp://ftp.inrialpes.fr/pub/vasy/traian/manual.ps.Z`, November 2000.
17. M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, and J. Visser. The ASF+SDF Meta-Environment: A Component-Based Language Development Environment. In *Proc. of CC '2001*, volume 2027 of *LNCS*, 2001.
18. P. van Eijk, A. Belinfante, H. Eertink, and H. Alblas. The Term Processor Generator Kimwitu. In *Proc. of TACAS '97*, 1997.