C-to-C: A Type-checking Preprocessor for C Extension Languages

Robert C. Miller*

rcm@theory.lcs.mit.edu

MIT Laboratory for Computer Science

C extension languages — programming languages which extend C with new keywords or syntax — have lately become popular in parallel and distributed computing research. Simple C extension languages can be translated into pure C by a macro preprocessor. More ambitious extensions, however, require a translator with features of a true compiler, like type checking and data-flow analysis. One familiar example is C++, which requires stronger type checking than C. A more recent example is Cilk, a multithreaded C extension language under development at MIT LCS [BJK+95]. In the course of writing a type-checking preprocessor for Cilk, I have developed C-to-C, a preprocessor framework that may help other extension language developers who want the benefit of a full compiler without the concomitant development and porting costs.

C-to-C parses a C program into an abstract syntax tree (AST) representation, performs type checking, data-flow analysis, and tree transformations directly on the AST, then unparses the AST to recover the original C program. To build a C extension language preprocessor based on C-to-C, a developer modifies the front-end to recognize the extension language, then adds transformations that convert extension syntax into ordinary C syntax.

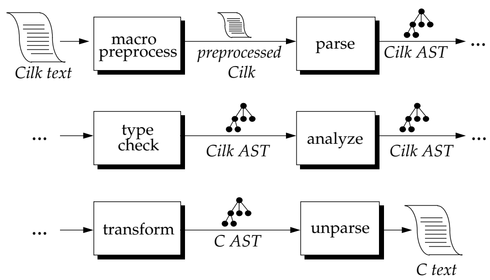

Figure 1: Translation phases of the Cilk preprocessor, which is based on C-to-C.

With C-to-C, writing a preprocessor is an attractive alternative to writing a full compiler. C-to-C encapsulates the hard work of ANSI compatibility done by Eric A. Brewer, Michael D. Noakes, and others who worked on the MIT Alewife compiler, so a language developer can concentrate on language extensions, not ANSI compatibility. In addition, a translator based on C-to-C is portable and extensible — highly desirable qualities for language research. Lastly, the output of a C-to-C translator is readable, high-level C, not "assembly language written in C" as might be expected.

A translator based on C-to-C is easier to port than a full compiler. C-to-C itself is written in ANSI C. Extension language features are translated into high-level C, which can be made portable. The Cilk preprocessor, for in-

*Rob Miller is advised by Charles E. Leiserson and is supported in part by ARPA contract N00014-94-1-0985. Address: Room 226, 545 Technology Square, Cambridge MA 02139.

stance, generates portable C which calls a runtime system, so moving Cilk to a new architecture entails no changes to the preprocessor whatsoever; only the relatively small runtime system must be ported.

A translator based on C-to-C is easier to extend than existing public-domain C compilers. Its lexical scanner and parser are automatically generated from `lex/yacc` specifications, which are easier to change than the hand-coded front-ends in [B+94, FH95, AALT95]. Type checking and data-flow analysis are table-driven, so that new statements and expressions can be specified with a few method functions. Transformations are expressed as operations directly on the AST, which we found easier to manipulate than the intermediate representation of compilers like `gcc` [S+]. Note that, unlike most compilers, C-to-C performs data-flow analysis directly on the abstract syntax tree, not on an intermediate, basic-block representation, so data-flow information refers to statements and expressions in the original source. The standard iterative algorithm [ASU88] implemented by C-to-C can be extended to solve any monotonic data-flow analysis problem, including live variables, reaching definitions, available expressions, and constant propagation.

Finally, a translator based on C-to-C produces output that is high-level, readable, and readily compared with its input, since only extension language features are translated. C-to-C preserves all pure C syntax, line numbering, indentation, and constant expressions used in the original program. The readability of C-to-C's output simplifies the process of checking that extension language constructs have been translated correctly.

C-to-C has been successfully used as the basis for the new Cilk preprocessor. The source code of C-to-C is freely available by anonymous ftp from `theory.lcs.mit.edu`, in directory `/pub/c2c`.

References

[AALT95] S. P. Amarasinghe, J. M. Anderson, M. S. Lam, and C. W. Tseng. The SUIF compiler for scalable parallel machines. In *Proc. of 7th SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[ASU88] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1988.

[B+94] François Bodin et al. Sage++: An object-oriented toolkit and class library for building Fortran and C++ restructuring tools. In *Proceedings of OONSKI*, 1994. Also available as `ftp://ftp.extreme.indiana.edu/pub/sage/oonski94.ps.gz`.

[BJK+95] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *Proc. of 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Santa Barbara, CA, July 1995. To appear.

[FH95] Chris Fraser and David Hanson. *A Retargetable C Compiler: Design and Implementation.* Benjamin/Cummings, 1995.

[S+] Richard Stallman et al. GNU C compiler (gcc) version 2.0. Source code and documentation available via `ftp://prep.ai.mit.edu/pub/gnu`.