The Compiler Compiler - Reflections of a User 50 Years On

George Coulouris*

Background

In 1962 I joined the London University Institute of Computer Science as an enthusiastic young research assistant. My appointment was linked to the impending arrival of the London Atlas and the perceived need by the senior staff, especially Dick Buckingham and Mike Bernal, for a suitable high-level language to support the development of a wider range of applications. Two teams were already working on language designs at the Institute and I chose to join the one working with a larger Cambridge team led by Christopher Strachey on the design of the CPL programming language [7]. A preliminary specification for the language had emerged in the form of several notes and it was suggested that I look into the construction of a compiler. The story of the CPL project and its eventual demise has recently been documented in David Hartley's excellent retrospective article [1].

I came to the project with some industrial programming experience but absolutely no knowledge of compiler construction. None of my colleagues was better-placed for the task of constructing a CPL compiler, so I scanned the horizon and the Compiler Compiler popped up – a revelation that was to have a major impact on my early professional life. (I have a recollection of attending a talk given in London by Tony Brooker at about that time, but cannot say whether that was the source of my first knowledge of the Compiler Compiler (henceforth abbreviated as *CC*). Publications by Brooker and his colleagues on the CC were yet to appear [5, 6]).

Textbooks on compiler construction - and indeed on most topics in computing - were non-existent at that time. (Don Knuth is reported [2] to have begun to write a book on compiler construction in 1962, only to realise that it would leave too many related important topics undiscussed - so he transformed the project into his epic 'Art of Computer Programming'.) But with the help of Tony Brooker and Derrick Morris's concise but instructive documentation and especially their examples, I was able to infer most of the relevant principles and get started on a project that was to occupy me for about two years.

The Compiler Compiler was an early but outstanding example of a domain-specific language. At their best, domain-specific languages give users a clear model of the application domain and a framework enabling them to get started with exploring implementation approaches, converging to a fully working implementation.

The construction of a top-down syntax-driven compiler is now well understood, but for me and probably quite a few others in those early years the CC provided the necessary understanding. One of those was Saul Rosen of Purdue University, who later wrote an excellent tutorial survey of the CC published in the Communications of the ACM in 1964

[3]. (That was well after my need to understand and use the compiler so I was unable to benefit from his efforts. But I have used his survey to refresh my memory for the brief summary of selected aspects below.)

Some interesting aspects of the CC

To build a compiler for a new language, one began by defining the syntax as a set of *format class definitions* in a notation that would now be recognised as a homologue of the Extended Backus–Naur notation. The notation was very well-conceived; it was easy to learn and apply and the CC included sufficient low-level primitives (called *basic formats*) to enable such things as numbers, variable names and so on to be efficiently recognised.

An *analysis routine* built-in to the CC would process a source program according to the given class definitions and generate an *analysis record* denoting the syntax tree for the source program in terms of the syntax supplied. The remaining and more difficult task of the compiler writer was to write a set of *format routines* that would generate executable code corresponding to the statements recognised in the source program. In the case of our CPL compiler we were aiming to generate Atlas assembly code - the use of byte code virtual machines as compiler targets was some years away.

The construction of format routines to perform the code generation was a classic system programming task requiring a deep and detail understanding of the target machine's architecture and several key data structures for use as symbol tables, memory maps and so on. The most surprising thing about the Compiler Compiler was that Tony Brooker and Derrick Morris had already encountered the requirements in the process of building the CC (and presumably in their earlier efforts to build compilers for other machines), and they had formulated good generalised representations for them.

Thus the notation used for writing format routines was essentially a system programming language of the same class as BCPL, C and so on. The CC was perhaps the first machine-independent language to explicitly include the notions of memory address, memory word, address arithmetic and indirection. Format routines were in fact executable analysis records. When a CC format routine was processed by the analysis routine the result was an executable tree structure containing references to other routines and markers denoting the parameters remaining to be inserted. To execute a routine a copy of its analysis record was made in a working area of memory similar to an execution stack and when values for all of the parameters have been substituted it is executed. This strongly analogous to modern interpretive language execution mechanisms. But Brooker and Morris were not content to leave all of the routines to be executed in the interpretive mode. They included a mechanism to translate format routines that did not contain parameters into assembly code for direct execution and this much improved the systems performance.

The CPL1 compiler

Work on a CPL compiler, dubbed the 'London CPL1 Compiler' began in mid-1963 and a working version was released in the autumn of 1964. Initial debugging of syntax class

definitions and an understanding of the format routine language were achieved well before the London Atlas had been delivered and commissioned through several visits to Manchester in the autumn of 1963, where the prototype Atlas was already available for use. Advice and tuition were freely and generously given by Derrick Morris - something I found essential to gaining an understanding of the workings of the CC.

The London CPL1 Compiler was restricted in two ways.

1. Program size was restricted to a couple of hundred lines.

   The nested block structure of CPL necessitated the analysis and initial processing of an entire source program before any code could be generated. Two-pass compilation would of course have been possible but would probably have required operator intervention, since we were still in the era of serial input devices. In retrospect, we ought to have bitten that bullet.

2. It had some semantic restrictions compared to the full CPL language.

   String variables and operations on them were implemented, but without the garbage collection that would have made them more effective and useful. Function closures were a pioneering feature of CPL that I fully appreciated only in the closing stages of the project (through informal interactions with Peter Landin at the legendary 'Mervyn Pragnell logic study group' that was running contemporaneously at the Institute). I failed to understand them sufficiently well to define an effective implementation.

Despite those restrictions the compiler enabled a small number of users to experience some of the benefits of CPL – a block structured procedural language with more extensive support for non-numeric data, functions and procedures than any other contemporary language. A paper on the London CPL1 compiler project, illustrated with some CC code fragments, was written and published in the Computer Journal in 1968 [4].

Reflections

The Compiler Compiler was remarkable in many ways. I still find it a quite amazing achievement in terms of the innovations that it contained and the effectiveness of its design and implementation. It encompassed innovative contributions at so many levels, from the very concept of a compiler-compiler to the inclusion of a domain-specific language for applications in system programming and the combination of interpretation with code generation.

Its implementation was completed at a time when we had virtually no debugging tools, with only paper tape for inputting programs, no file storage and certainly no capability for interactive debugging or user interaction. What a *tour de force*!

References

1. David Hartley (2012), CPL - failed venture or noble ancestor?, *IEEE Annals of the History of Computing*, IEEE computer Society Digital Library. IEEE Computer Society, <http://doi.ieeecomputersociety.org/10.1109/MAHC.2012.37>

2. Wikipedia article: 'The Art of Computer Programming', http://en.wikipedia.org/w/index.php?title=The_Art_of_Computer_Programming&oldid=515539798 (downloaded October 2012).

3. Saul Rosen (1964), A Compiler Building System Developed by Brooker and Morris, Communications of the ACM Vol. 7, No. 7

4. Coulouris G.F, Goodey T.J., Hill R.W., Keeling R.W. and Levin D, (1968) The London CPL1 Compiler, Computer Journal Volume 11, Issue 1, pp. 26-30.

5. Brooker R A , MacCallum I R, Morris D and Rohl J S, (1963) *The Compiler Compiler.* Annual Review of Automatic Programming Vol 3, 1963, p 229. Pergamon, London.

6. Brooker, R A, Morris, D and Rohl, J S, (1967) Experience with the Compiler Compiler. Computer Journal, vol. 9, p 345.

7. Barron, D.W., Buxton, J.N., Hartley, D.F., Nixon, E. and Strachey, C . (1963). The Main features of CPL. Computer Journal, vol.6, p 134.

* George Coulouris joined the Institute of Computer Science, London University in 1962. In 1965 on completing the project described in this note he became a lecturer at Imperial College and in 1971 joined a fledgling Department of Computer Science at Queen Mary College, where he remained until his retirement in 1998, holding posts of Lecturer, Reader and Professor of Computer Systems. In 1988 he co-authored and published the first comprehensive textbook on distributed systems, now in its fifth edition and still frequently adopted for undergraduate courses.

Since retirement George has spent time at the Cambridge University Computer Laboratory working with a research group focussed on ubiquitous and sentient systems.