XPL

From Wikipedia, the free encyclopedia

This article is about a dialect of the PL/I programming language. For the meaning of the term and other uses, see XPL (disambiguation).

XPL is a programming language based on PL/I, and a portable one-pass compiler written in its own language, and a parser generator tool for easily implementing similar compilers for other languages. XPL was designed in 1967 as a way to teach compiler design principles and as starting point for students to build compilers for their own languages.

XPL was designed and implemented by William McKeeman and David B. Wortman at University of California, Santa Cruz and James J. Horning and others at Stanford University. XPL was first announced at the 1968 Fall Joint Computer Conference. The methods and compiler are described in detail in the 1971 textbook *A Compiler Generator*.

They called the combined work a 'compiler generator'. But that implies little or no language- or target-specific programming is required to build a compiler for a new language or new target. A better label for XPL is a Translator Writing System. It helps you write a compiler with less new or changed programming code.

Contents

Language

The XPL language is a simple, small, efficient dialect of PL/I intended mainly for the task of writing compilers. The XPL language was also used for other purposes once it was available. XPL can be compiled easily to most modern machines by a simple compiler. Compiler internals can be written easily in XPL, and the code is easy to read.

The PL/I language was designed by an IBM committee in 1964 as a comprehensive language replacing Fortran, COBOL, and ALGOL, and meeting all customer and internal needs. These ambitious goals made PL/I complex, hard to implement efficiently, and sometimes surprising when used. XPL is a small dialect of the full language. XPL has one added feature not found in PL/I: a STRING datatype with dynamic lengths. String values live in a separate text-only heap memory space with automatic garbage collection of stale values. Much of what a simple compiler does is manipulating input text and output byte streams, so this feature helps simplify XPL-based compilers.

Components

XCOM

The XPL compiler, called **XCOM**, is a one-pass compiler using a table-driven parser and simple code generation techniques. Versions of XCOM exist for different machine architectures, using different hand-written code generation modules for those targets. The original target was IBM System/360, which is a proper subset of IBM System/370, IBM System/390 and IBM System z.

XCOM compiles from XPL source code, but since XCOM itself is written in XPL it can compile itself – it is a *self-compiling compiler*, not reliant on other compilers. Several famous languages have self-compiling compilers, including Burroughs B5000 Algol, PL/I, C, LISP, and Java. Creating such compilers is a chicken-and-egg conundrum. The language is first implemented by a temporary compiler written in some other language, or even by an interpreter (often an interpreter for an intermediate code, as BCPL can do with intcode or O-code).

XCOM began as an Algol program running on Burroughs machines, translating XPL source code into System/360 machine code. Someone manually turned its Algol source code into XPL source code. That XPL version of XCOM was then compiled on Burroughs, creating a self-compiling XCOM for System/360 machines. The Algol version was then thrown away, and all further improvements happened in the XPL version only. This is called bootstrapping the compiler. The authors of XPL invented the Tombstone diagram or T-diagram to document the bootstrapping process.

Retargeting the compiler for a new machine architecture is a similar exercise, except only the code generation modules need to be changed.

XCOM is a one-pass compiler (but with an emitted code fix-up process for forward branches, loops and other defined situations). It emits machine code for each statement as each grammar rule within a statement is recognized, rather than waiting until it has parsed the entire procedure or entire program. There are no parse trees or other required intermediate program forms, and no loop-wide or procedure-wide optimizations. XCOM does, however, perform peephole optimization. The code generation response to each grammar rule is attached to that rule. This immediate approach can result in inefficient code and inefficient use of machine registers. Such are offset by the efficiency of implementation, namely, the use of dynamic strings mentioned earlier: in processing text during compilation, substring operations are frequently performed. These are as fast as an assignment to an integer; the actual substring is not moved. In short, it is quick, easy to teach in a short course, fits into modest-sized memories, and is easy to change for different languages or different target machines.

ANALYZER

The XCOM compiler has a hand-written lexical scanner and a mechanically-generated parser. The syntax of the compiler's input language (in this case, XPL) is described by a simplified BNF grammar. XPL's grammar analyzer tool **ANALYZER** or **XA** turns this into a set of large data tables describing all legal combinations of the syntax rules and how to discern them. This table generation step is re-done only when the language is changed. When the compiler runs, those data tables are used by a small, language-independent parsing algorithm to parse and respond to the input language. This style of table-driven parser is generally easier to write than an entirely hand-written recursive descent parser. XCOM uses a bottom-up parsing method, in which the compiler can delay its decision about which syntax rule it has encountered until it has seen the rightmost end of that phrase. This handles a wider range of programming languages than top-down methods, in which the compiler must guess or commit to a specific syntax rule early, when it has only seen the left end of a phrase.

Runtime

XPL includes a minimal **runtime support library** for allocating and garbage-collecting XPL string values. The source code for this library must be included into most every program written in XPL.

SKELETON

The last piece of the XPL compiler writing system is an example compiler named **SKELETON**. This is just XCOM with parse tables for an example toy grammar instead of XPL's full grammar. It is a starting point for building a compiler for some new language, if that language differs much from XPL.

XMON

XPL is run under the control of a monitor, **XMON**, which is the only operating system-specific part of this system, and which acts as a "loader" for XCOM itself or any programs which were developed using XCOM, and also provides three auxiliary storage devices for XCOM's use, and which are directly accessed by block number. The originally published XMON was optimized for IBM 2311 disk drives (ca. 1964), and was most efficient on those drives. XMON is about 50 percent efficient on IBM 2314 disk drives (ca. 1965), and is significantly less efficient on subsequently introduced disk drives, such as the IBM 3330 (ca. 1970), 3330-11 (ca. 1973) and 3350 (ca. 1975), and is dramatically less efficient on devices with much larger larger track capacities, such as the IBM 3390 (ca. 1989).

Converting XMON from its primitive use of NOTE, POINT and READ/WRITE disk operations (with precisely 1 block per track, with the entire remainder of the track being erased, hence wasted space) to EXCP and XDAP (with n blocks per track, where n is computed from the target device's physical characteristics and can be significantly greater than 1, and with no wasted space) yields a dramatic increase in disk utilization efficiency, and a corresponding reduction in operating system overhead.

Although originally developed for OS/360, XMON (either the original NOTE, POINT and READ/WRITE implementation; or the EXCP and XDAP enhancement) will run on subsequently released IBM OSes, including OS/370, XA, OS/390 and z/OS, generally without any changes.

Parsing

XCOM originally used a now-obsolete bottom-up parse table method called *Mixed Strategy Precedence*, invented by the XPL team (although the *officially* released version retains the MSP parser and *does not* include later-released "peephole optimizations" and additional data types which were developed outside of the *original* implementation team.) MSP is a generalization of the simple precedence parser method invented by Niklaus Wirth for PL360. Simple precedence is itself a generalization of the trivially simple operator precedence methods that work nicely for expressions like A+B*(C+D)-E. MSP tables include a list of expected triplets of language symbols. This list grows larger as the cube of the grammar size, and becomes quite large for typical full programming languages. XPL-derived compilers were difficult to fit onto minicomputers of the 1970s with limited memories.[1] MSP is also not powerful enough to handle all likely grammars. It is applicable only when the language designer can tweak the language definition to fit MSP's restrictions, before the language is widely used.

XCOM and XA were subsequently changed to instead use a variant of Donald Knuth's LR parser bottom-up method.[2] XCOM's variant is called Simple LR or SLR. It handles more grammars than MSP but not quite as many grammars as LALR or full LR(1). The differences from LR(1) are mostly in the table generator's algorithms, not in the compile-time parser method. XCOM and XA predate the widespread availability of Unix and its yacc parser generator tool. XA and yacc have similar purposes.

XPL is open source. The System/360 version of XPL was distributed through the IBM SHARE users organization. Other groups ported XPL onto many of the larger machines of the 1970s. Various groups extended XPL, or used XPL to implement other moderate-sized languages.

1. Indeed, using a hand-written LALR-like analyzer and a particularly efficient "decomposition" procedure for the produced parsing tables, it was possible to generate a parser for the entire XPL language on a 2 MHz Z80 microcomuter which had only 48 kilobytes of internal memory (DRAM) and only 100 kilobytes of external memory (floppy disk) running under CP/M. This version was completed in 1980. Porting to MacOS (9, later X) was subsequently completed.
2. This version was NOT released to the general community, hence it remains proprietary to its authors, or to their institutions. Repeated requests for an SLR(1) or an LALR(1) distribution of XPL have been ignored by its authors.

Applications

XPL has been used to develop a number of compilers for various languages and systems.

- Stony Brook Pascal
- HAL/S, the language used for the Space Shuttle program
- MALUS, a system programming language used by General Motors to develop their Multiple Console Time Sharing System
- New England Digital used a variant of XPL, called "Scientific XPL" for their ABLE series computers, used for laboratory automation, computer networking, and control of music synthesis hardware, starting in the mid-1970's

Current status

XPL continues to be ported to current computers. An X86/FreeBSD port was done in 2000.[1]

References

- McKeeman, William Marshall; Horning, James J.; and Wortman, David B., *A Compiler Generator* (1971), ISBN 978-0-13-155077-3. The definitive reference.

1. Bodenstab, Dave. "Dave Bodenstab's Home Page". Retrieved Feb 6, 2015.

Bibliography

- Alexander, W.G. and Wortman, D.B. "Static and Dynamic Charactersistics of XPL Programs." IEEE Computer Nov 1975; 41-46.
- Ancona, Massimo, Dodero, Gabriella, and Durante, Ercole Luigi "Cross software development for microprocessors using a translator writing system" Proceedings of the 4th International Conference on Software Engineering 1979: 399-402.
- Kamnitzer, S.H. "Bootstrapping XPL from IBM/360 to UNIVAC 1100." ACM SIGPLAN Notices May 1975: 14-20.
- Karger, Paul A. "An Implementation of XPL for Multics." SB thesis. Massachusetts Institute of Technology, 1972.
- Klumpp, Allan R. "Space Station Flight Software: Hal/S or Ada?" Computer March 1985: 20-28.
- Leach, Geoffrey and Golde, Helmut. "Bootstrapping XPL to an XDS Sigma 5 Computer." Software Practice and Experience 3 (1973): 235-244.
- McKeeman, William M., Horning, James J. and Wortman, David B. A Compiler Generator. Englewood Cliffs, NJ: Prentice-Hall, 1970.
- McKeeman, W. M., Horning, James J., Nelson, E.C., and Wortman, D. B "The XPL compiler generator system." AFIPS Conference Proceedings: 1968 Fall Joint Computer Conference. Washington DC: The Thompson Book Company. 1968: 617-635.
- Sitton, Gary A., Kendrick, Thomas A., and Carrick, jr., A. Gil. "The PL/EXUS Language and Virtual Machine" Proceedings of the ACM-IEEE Symposium on High-level-language Computer Architecture Nov, 1973: 124-130.
- Slimick, John "Current Systems Implementation Languages: One User's View" Proceedings of the SIGPLAN symposium on Languages for system implementation Oct, 1971: 20-28.
- Storm, Mark W., and Polk, Jim A. "Usage of an XPL Based Compiler Generator System" Proceedings of the 14th annual ACM Southeast Regional Conference Apr, 1976: 19-26.
- Wortman, D.B. "A roster of XPL implementations." ACM SIGPLAN Notices Jan 1978: 70-74.

External links

- The XPL Programming Language
- *A Compiler Generator* page at Amazon.com
- Scientific XPL for New England Digital Corporation's ABLE Series Computers