BABEL, AN APPLICATION OF EXTENSIBLE COMPILERS

R. S. Scowen
National Physical Laboratory
Teddington, Middlesex, England

INTRODUCTION

The normal approach in providing an extensible programming language seems to be to design and implement a base language which has facilities enabling the programmer to define and use extensions. This paper discusses a solution using an alternative approach in which extensions are made by changing the compiler. Of course, in theory, any compiler can be altered (it is only a computer program); in practice it is probably not so easy since difficulties will arise if the compiler has been designed for one fixed standard language on one particular computer. If a compiler is to be altered, it must possess various properties; there should be no danger of accidentally invalidating an existing program, and it must be clear what changes are required to make a desired extension. It is also desirable that only a small number of changes should be required to make an extension.

The method of writing compilers which is described below satisfies these conditions and has been adopted in three different applications:-

(1) Babel, a conventional high level programming language.

(2) SOAP, a program which documents and edits an ALGOL 60 program.

(3) PL516, a high level assembly language for the Honeywell DDP516 computer.

In each of these systems there is a processor which reads a program as input, analyses its syntax, and produces output which depends on the syntax. Each processor has the same overall structure, and can easily be extended or altered to cope with a different input language or to process a language in a different way.

This paper describes Babel giving details of the system, the structure of the compiler and some of the extensions which have been made. SOAP has been described in Scowen 1971 and PL516 has been described in Wichmann 1970, and Bell and Wichmann 1971.

THE BABEL LANGUAGE

The Babel (*1) language is a cleaner version of ALGOL 60. Concepts which are only applicable in a local context have been removed from the language,

(1) The name Babel is not an acronym; it is meant as a perpetual warning of the fate awaiting anyone who makes too many extensions.

e.g. own, switch, implicit label declarations, implicit call-by-name.

Other more useful concepts have been added:- (1) Cases, see Hoare 1964, (2) While and Until statements, (3) Extra global entities, including some for Input Output, (4) Constant valued entities, (5) More operators, e.g. ABS, SIGN, REM.

All these changes are designed to help the programmer by making a short and simple program an efficient one.

The rules of the language have been formulated so that there are fewer exceptional cases than in ALGOL 60; for instance, in Babel, one sort of statement is always syntactically equivalent to any other form of statement. Other changes have been made so that the compiler can check as much as possible of a program during translation.

The Babel compiler

The Babel compiler is written in ALGOL 60; this has several very important advantages (see Garwick 1966):-

(1) It provides a formal definition of the language.

(2) It is machine independent.

(3) The debugging has been simplified.

(4) It is easier to document and understand the compiler.

(5) It has helped to ensure that the structure remains modular, neat and tidy.

The ALGOL version is naturally too slow to be a practical tool, but from it a more efficient compiler has been produced. D. Schofield used a Compiler - Compiler (see Brooker et al 1963) to convert most of the translator into a KDF9 assembly language. Other parts of the compiler were also converted to assembly language automatically. The remaining parts of the new compiler are either software that already existed (i.e. for input/output, standard functions) or were coded by hand using the ALGOL compiler as a blueprint. This compiler has a satisfactory performance (compiling a page of text per sec. and producing 600 bytes of code per sec.). The speed of translation is approximately equal to that of the KDF9 Whetstone ALGOL Compiler (Randell and Russell, 1964) which was specially designed to translate rapidly.

The compiler contains several segments:-

(1) An Input section which reads the text of a Babel program and splits it into basic symbols.

(2) A Translator section which analyses the syntax of a Babel program and translates it into the machine code of a Delta computer (a hypothetical machine with a Reverse Polish order code).

(3) A Listing section which prints the Delta machine code of a

translated program.

(4) An Interpreter section which simulates a Delta computer and interprets the code produced by the Translator.

(5) A Conversion section which turns the Delta code into KDF9 machine code which can be executed directly.

Many of the details have been described in a report (Scowen 1970).

The Babel translator

Most of the compiler segments have a straightforward form of construction which is simple to extend, but the Translator has had to be designed in a special way so that facilities can be added or deleted from the language.

The Translator is modular so that any new concept affects the compiler in clearly defined parts of it. Each syntactic part of the language corresponds to a unique part of the Translator and each module of the Translator is a routine for a different part of the Babel language: thus one routine translates a block, another translates an expression and there are others to translate a constant, a variable, a conditional, etc.

Each routine is an ALGOL procedure, which, when called, makes a single pass through the text of the appropriate syntactic part of the Babel program. A routine reads the basic symbols one at a time, translating as it reads them and calling other routines whenever appropriate. The routines call each other recursively in a way determined by the syntactical structure of the program being translated.

The routines have been written so that each one can be re-written easily as a subroutine in an assembly language; some of the properties which ensure this are:-

(1) The body of each routine is a compound statement, not a block.

(2) The routines do not have any parameters; instead they communicate with each other using a (fairly small) number of global entities. Whenever necessary the current value of one of these entities is preserved on a stack.

Any version of Babel is a superset or a subset

It would be most unfortunate if a new version of Babel invalidated an existing program. The modular single pass structure of the Translator is the most important feature which makes it simple to ensure that an extension to Babel creates a superset and deleting a facility creates a subset. In either case programs which do not use the changed feature compile and run with the same effect as previously. Other features are:-

(1) There is a clear distinction between basic symbols and identifiers; a programmer creates identifiers and the extender of Babel can create new basic symbols.

(2) There are no implicit type conversion operators so that the introduction of a new type cannot affect existing programs.

(3) Global entities are regarded as being declared in a block which surrounds the program and not in the outermost block of the program.

Note also that the modular nature of the Translator makes it far easier to implement a general extension than a number of special cases. A general extension can often be made by writing a few statements which call one or more of the existing routines, but each special case would have to programmed separately. For example, there are only two reasonable possibilities for a subscript, either an integer constant or any integer expression.

The strategy of extension

At first sight it may seem that extending a language by altering the compiler is an inferior and less powerful tool than being able to specify a language extension in a program. It is true that an ordinary programmer will not be able to define a private extension and there is a danger of an unmanageable number of compilers, but there are some advantages:-

(1) It is possible to define extensions which cannot be expressed simply in terms of the original language.

(2) It is possible to define a subset or to change the meaning of the language.

(3) It does not matter whether a programmer uses any of the possible extensions or not; no noticeable extra time is needed for either compilation or execution.

(4) The compiler is simple and fast.

(5) Extensions can be made which do not change the language but instead give better diagnostics (e.g. a sensible post mortem), or document programs (e.g. SOAP), or provide facilities for measuring program performance and analysis.

(6) Error messages can be given in terms of the full language rather than the base language.

(7) The user has less to learn.

Finally there is no need to teach a large number of people how to make extensions and thus there is less danger of misuse. All extensions can be implemented efficiently because they are made jointly by a software engineer who amends the compiler and the customer who knows what he wants but not how to express it. As a result it is hoped that new concepts will always be implemented in such a way that the programmer is guided into expressing his algorithm in a sensible way that is both natural and efficient.

BABEL EXTENSIONS

A new form of statement

A decision table provides a compact method of specifying many complicated algorithms which cannot be expressed clearly withh case and conditional statements. A common method of implementation is to write a preprocessor which will convert a decision table into a procedure in a standard programming language which is then inserted into the rest of the user's program. In Babel an extension has been made so that a decision table is merely a new form of statement in the language. This is more convenient and efficient than having to use a preprocessor.

New types

It is possible to introduce a new type into Babel. If it is an 'assignment type' (i.e. values of the type can be assigned to variables), then some facilities become available as soon as the new basic symbols can be recognized. With the new type it will be possible to create and use variables, arrays, functions, parameters to procedures, assignment statements, conditional and case expressions. This is because the parts of the Translator which compile these constructions work for any 'assignment type'.

What else has to be done? It will be necessary to define a constant of the new type, both how it is to be represented in a program, in a data file, and in the computer during the execution of a program. Probably it will also be desirable to define some new operators; for each one it is necessary to specify its name, precedence, the type of the result, and the code to be generated. The extension is completed by defining any new standard procedures and functions which are required, e.g. for input or output.

By creating new types it has been possible to create facilities for complex arithmetic and for processing text (as in compilers and editors). Another possibility is an extension for performing real fixed-point arithmetic; this would be desirable on small computers with no floating point hardware.

New structures

Record classes (see Wirth and Hoare 1966, Hoare 1968) have also been implemented as an extension to Babel. In this extension RECORDCLASS is a type and REFERENCE(recordclass identifier) is an 'assignment type'.

A record class declaration creates a record class and specifies its name, size and structure. A reference entity points to a record of one particular record class; any field of a record can be used as a variable or primary because the concept of a variable has been extended to include 'reference variable.field identifier'. The implementation has not been made with full generality because (as is well known) an unlicensed use of reference (or pointer) variables lets the programmer make obscure and catastrophic errors. In Babel a reference entity always points to a record of one particular record-class which exists at least as long as the reference entity. The compiler produces a translation error if the programmer tries to assign a reference entity to anything other than a record of the appropriate record class. The translator also checks comparisons and procedure parameters to see that they are compatible.

Other sorts of structure can easily be created besides simple variables, arrays and record classes. For instance it may occasionally be useful to have symmetric, band or sparse matrices; in each case the programmer would use the array as though it were stored in full and the compiler would generate code to calculate the appropriate address.

Syntax oriented documentation

A radical but simple alteration is to convert the Babel compiler so that it documents a Babel program in the same way as SOAP edits an ALGOL program. This is accomplished by amending the Babel Translator so that instead of compiling the program and outputting code, it outputs the text of the program. The original editing characters will have been discarded by the input section of the compiler and the Translator is extended so that new editing characters are output at points determined by the syntax of the program. Virtually the only other change which is necessary is to arrange to copy comments instead of discarding them in the Input section.

It is also fairly simple to modify the Translator so that it uses the name list to produce a table which gives a complete cross reference listing of all the entities in the program; i.e. the position of the declaration, assignments and evaluations of a variable, calls of the procedures and functions, etc..

Diagnostic aids

Another obvious possibility is to keep the language unchanged but to alter the code which is compiled. By this means it has been possible to extend Babel to provide diagnostic and program analysis aids. When a Babel program fails, the compiler not only indicates the sort of error but also prints a retroactive trace to show what was happening just before the error. The compiler also prints a postmortem which clearly specifies the name, type and current value of all the entities that existed when the program failed. Subscript checking can be made a translation option. Code can also be output to count the number of times each procedure is called and label passed; this table can then be printed with a listing of the program to show how frequently different parts were executed.

Other simple changes

Many other simple changes are possible; it is almost a trivial matter to create a new operator or global entity. The only difficulty that may occur is in deciding what code to execute when the operator or global entity is used. One new operator which has been provided gives the remainder after integer division. Two new global entities enable the programmer to create a file with his results instead of outputting it directly.

Conclusions and Acknowledgements

The implementation of Babel has shown that extensible compilers are practicable if the compiler has a modular structure based on the language. A wide range of extensions have been made without affecting existing programs. It is also considered that writing a prototype system in ALGOL 60 had many advantages because it made it easier to verify the ideas and to debug the system.

I am grateful to D. Allin, D. Schofield, M. Shimell and B. A. Wichmann for many discussions and assistance during the the design and implementation of Babel.

REFERENCES

D. A. Bell and B. A. Wichmann, An ALGOL-like Assembly Language for a Small Computer, Software - Practice and Experience, pp61 - 72, Vol 1, 1971.

R. A. Brooker, I. R. MacCullum, D. Morris, J. S. Rohl, The Compiler Compiler, in 'Annual Review in Automatic Programming', Vol 3, Pergamon, 1963.

J. V. Garwick, The definition of programming languages by their compilers, in 'Formal Language Description Languages for Computer Programming', North Holland, pp139 - 147, 1966.

C. A. R. Hoare, Case expressions. ALGOL Bulletin 18.3.7, Oct 1964.

C. A. R. Hoare, Record Handling in 'Programming Languages' edited by F. Genuys, Academic Press, pp291 - 348, 1968.

B. Randell and L. J. Russell, ALGOL 60 Implementation, Academic Press, 1964.

R. S. Scowen, The Babel Compiler, NPL CCU Report No 10, March 1970.

R. S. Scowen, D. Allin, A. L. Hillman, M. Shimell, SOAP - A program which documents and edits ALGOL 60 programs, Comp J, pp133 - 135, Vol 14, No 2, 1971.

B. A. Wichmann, PL516, An ALGOL-like Assembly Language for the DDP-516, NPL CCU Report No 9, Jan 1970.

N. Wirth and C. A. R. Hoare, A contribution to the development of ALGOL, Comm ACM, pp413 - 432, Vol 9, No 6, 1966.