PS-algol: an Algol with a Persistent Heap.

Malcolm Atkinson, Ken Chisholm, Paul Cockshott
Department of Computer Science, University of Edinburgh, Scotland

Keywords

Algol, heap, pointer, database.

Abstract

PS-algol is a dialect of algol for the programming of problems that would normally require a database management system. It supports a persistent heap, and an associative store; it has embedded within the language features to support tasks normally carried out by filing systems or database management systems.

1 Databases and programming languages

Database systems and programming languages have each developed their own method of structuring data. Database systems have developed relational, hierarchical, network and functional models of data structure [8 13 17 19]. Programming languages have, instead, vectors, multidimensional arrays, structures, access or reference variables and classes or abstract types as data structuring facilites. Given the data structuring facilities of programming languages the database models of data may be synthesised.

Programming languages are distinguished from database systems in their management of data primarily by the fact that the data objects created in the course of a program run do not persist beyond the run of the program. In a database system data objects persist beyond and between the runs of individual application programs. In order to handle persistent data, programming languages have relied upon file I/O or an interface to an external DBMS. This is discussed elsewhere [2 5].

In general, database systems have been interfaced to a programming language, by adding ad hoc extensions to the syntax of the programming language to acommodate the data model used in the external database system. Examples are the Codasyl extensions to Cobol and Fortran, or the UDL extension to PL/I [9 10].

Recently some languages have been designed with specific facilities making them suitable for data intensive applications [4 12 16 18 20]. Typically these add the data types of a predefined database model to a pre-existing collection of types. However only PS-algol and ELLE [1] handle persistent and transitory data uniformly. By uniformly we mean that any operation or type constructing facility of the host language is applicable to all data - whether persistent or transitory.

2 The hypothesis of minimum change

We have hypothesised that it should be possible to change an advanced algol into a database language without any change to its syntax.

By an advanced algol we mean a block structured algorithmic language with type definition facilities, a heap and access variables. Algol68 [22] and Pascal are examples of such languages. The minimum changes necessary to convert an advanced Algol into a database language are:

1. A persistent heap.

2. A method of linking programs with preserved heaps.

3. A mechanism to delimit transactions.

An additional desirable facility is:

4. An associative store.

A persistent heap is one on which a data structure built in one run of a program may be preserved to be used in other runs of the same or other programs. A method of identifying these preserved heaps and binding to them is then necessary. A transaction mechanism makes it possible to ensure the integrity of data. It ensures that all the changes made during a transaction are effected or that the data is restored to its original state [7].

Arrays indexed by scalar types provide an updateable mapping from a range of a discrete type to some other type. Associative store is a further generalisation of arrays. It allows the construction of updateable functions sparsely indexed by other types such as strings. Although associative store is not necessary in a database language, since it can be emulated in software, its effect is required often enough to make its inclusion desirable.

3 S-algol: the example language

S-algol is a language developed at St Andrews University, Scotland [14, 15]. It belongs to the Algol tradition, and has a very concise and orthogonal syntax. The guiding tenet of its design was that power is gained through simplicity and simplicity through generality. In expressive power and orthogonality it is somewhat above Pascal. Its most important features are its data structuring facilities.

The base types of the language are integer, boolean, real, file, string and pointer. Strong typing is ensured by a combination of compile time and run-time checks. Strings support the operations of concatenation and substring selection.

The type constructors are vector, and structure. Vectors are updateable mappings from dynamic ranges over the integers to one of the base types, or to a vector type. Multidimensional arrays may be formed by composing the vector construction operations. Structure classes are ordered cartesians of named fields belonging to one of the base types, or to a vector type.

Pointers are access descriptors which can reference instances of any of the structure classes but which may not reference instances of base types or vectors. The language provides the operations of field subscription and run-time type verification on pointers. This arangement makes it possible to write algorithms to manipulate pointers without the need to know the type of the referend.
The program fragment:

```
structure cons( pntr hd,tl)
structure string.atom( string val)
structure nil
let a.string  = " Dumpty"
let a.pointer = string.atom( "Humpty" ++ a.string )
let another.pointer := cons( a.pointer, nil )
while another.pointer isnt nil do
begin
        write if another.pointer(hd) is string.atom
              then  another.pointer(hd,val)
              else  "Not a string"
        write newline
        another.pointer := another.pointer ( tl )
end
write "End of list"
```

would produce output:

```
Humpty Dumpty
End of list
```

Note the following:

1. Identifier declarations are introduced by the word 'let' and are initialising.

2. The type of an identifier is given by the type of its initialising value.

3. Constant identifiers are initialised using '=', variables using ':='.

4. The operators 'is' and 'isnt' are used to check the class currently referenced by a pntr.

5. The declaration of a structure class implicitly declares a generator function of the same name, whose application yields an instance of the class.

The Heap

In S-algol all compound data objects: strings, structures, and vectors are generated on the heap. There is a garbage collector which preserves objects reachable from identifiers currently in scope.

Implementations

S-algol has currently been implemented on the following sytems.

Machine	Operating System
PDP11	Unix
VAX	VMS
Z80	CP/M
PE3220	Mouses

4 PS-algol as an extension to S-algol

PS-algol (Persistent S-algol) is a derivative of the language S-algol. Syntactically it is identical to S-algol. The only visible extension to the language is the addition of a number of new predeclared procedures and a predeclared type. The effect of these, however, is to greatly increase the power and generality of the S-algol heap. These procedures provide:

1. A method of associating a program heap with a named database.

2. A method of commiting a transaction upon the database thus simultaneously updating the database and preserving a subset of data on the heap.

3. An associative store.

4. A universal nil pointer.

The procedures are as follows

```
procedure open.database( string user.name, password, db.name->bool )
```

If the correct user password is supplied, this opens a database belonging to the specified user, with the db.name given. If the database does not yet exist it is created. If another user currently has a transaction open on the database then the open fails. Any failure is indicated by returning false otherwise a mapping is established between the database and the program's heap and true returned.

```
procedure root.table( ->pntr )
```

This procedure returns a pointer to an instance of the predefined class 'table' which provides an associative store facility. The internal structure of this class is hidden from programmers using PS-algol (in fact they are implemented as B-trees [6]) . However, the procedures that follow allow operations on instances of this class. Associated with each database there is a distinguished table that is returned by root.table.

```
procedure nil (-> pntr )
```

S-algol has no predefined nil pointer, so one generally declares a local nil pointer and uses it to designate the end of lists etc. The weakness of this in a persistent environment is that there would no longer be a unique nil pointer, but separate ones for each program that ran against the database. This procedure returns a system wide nil pointer.

procedure enter(string key,pntr table,value)

This enters the parameter value into the table using the key. The value, being a pointer, can reference any arbitrarily complex data structure, including another table. The effect of entering a value/key pair is to set up an association between the key and the value, allowing the key to be used for later retrieval of the value. If the value is nil the effect is that of deleting the item from the table.

procedure lookup(string key, pntr table -> pntr)

This returns the value last associated with the key in the specified table. If no value has been associated with this key, nil is returned.

procedure table(->pntr)

This returns a pointer to an empty table.

procedure scan(pntr table,(string->bool)user)

This scans the table with the function `user' which takes a string parameter, invoking it once for every key in the table or until it returns false. When the procedure exits, each key in the table will have been provided as a parameter to `user' once. By means of side effects the user procedure can gather statistics about, print or modify the entries in the table.

procedure commit

This causes all the objects which can be reached from the root table to be saved in the database and then terminates the program. If the program terminates without invoking commit, none of the changes made to the database during the run of this program will take effect.

5. Implementation

PS-algol has been implemented on a Perkin Elmer 3220 under the Mouses operating system [21]. It is based upon the 3220 S-algol system.

It was not necessary to modify the compiler. The additional built in procedures and the predefined class Table were added to the language by modifying the compiler`s prelude file. The most important modification was to the run-time system. The original S-algol run-time system consisted of a heap management system and a library of predefined procedures. The PS-algol run-time system was extended to include:

1. Additional library procedures.
2. A persistence manager.
3. A commit module.
4. A new garbage collector.

These new modules communicate over an interprocess communications system with a data base management process called the Chunk Mangement System (CMS) [3]. Their functions can best be understood by examining

what happens when a PS-algol program is executed twice.

a) The program starts and calls "open.database" for the first time. This operates by remote procedure entry. A message to the CMS, invokes a procedure in the database management system to create a new data base.
b) The program builds a data structure on the heap. Garbage collections occur as in S-algol.
c) The program builds structures as usual on the heap, and inserts pointers to some of these in the root table. The tables are created and manipulated as B-trees in the CMS [6]. Pointers to tables are returned as Persistent Identifiers (PIDs). PIDs are tokens to database objects. They are distinguished from local pointers.
d) The program terminates by calling commit. All items on the heap reachable from the root table are sent back to the CMS. All new objects sent back are given PIDs. All pointers in objects sent back are overwritten with the PIDS of the objects they pointed to. This task is done by the commit module.
e) Another program starts against the same database. Using the routine lookup, it obtains a pointer to an object previously stored in the root table. This pointer will be a PID sent over the message interface.
f) The program attempts to dereference the PID. The fact that it is a PID is detected by the dereference operator and the persistence manager is invoked. The persistence manager invokes a remote procedure in the CMS to fetch the requested item from the data base. The pointer variable that was derefenced is over-written with the local address of the item which is now on the heap, and the dereferencing is retried. This may occur repeatedly as items brought from the database may contain pointer fields which are initially PIDS. Any attempt to dereference these will be similarly trapped.
g) The program commits and the updated data structure is written back as in (d).

6. Commentary

We have explored the use of a persistent heap in a traditional block structured language. We have demonstrated, by building a working system that it is feasible to provide such a language on the basis of a minimal modification of an advanced Algol. We believe that this approach can be supported efficiently. Two methods of support are being evaluated: one in which the run-time environment is intimately connected to the database manager; the other in which database management is centralised on a database server and varying amounts of the local memory management and dereferencing algorithms are executed on the client machines.

At present the system does not totally enforce type protection between programs. Methods of providing type protection, the independent development of programs and the provision of concurrent access are all worthy of investigation.

We believe that this language would be suitable to directly implement more traditional database systems. It is also likely to provide a more productive development environment for quite extensive classes of programs performing data manipulation.

We consider this language a prototype and plan to investigate how the same extensions should be added to Ada [11].

References

[1] Albano, A., Occuchiuto, M.E., Orsini, R. A uniform management of temporary and persistent complex data in high level languages. Nota Scientifica, S-80-15, September 1980

[2] Atkinson, M.P. Database systems and programming languages. In: Proceedings of the Fourth International Conference on Very Large Databases, September 1978, 408-19

[3] Atkinson, M.P., Chisholm, K.J., Cockshott, W.P., The Chunk Management System. University of Edinburgh, Department of Computer Science, Internal Report - To Appear.

[4] Atkinson, M.P., Chisholm, K.J., Cockshott, W.P., The New Edinburgh Persistent Algorithmic Language. in Databases, Pergammon Infotech State of the Art report, series 9, no 8, Pergammon Infotech, Maidenhead, England, December 1981.

[5] Atkinson, M.P., Martin, J.A., and Jordan, M.J. A uniform modular structure for databases and programs. in Data Design, Infotech State of the Art report, series 8, no 4, Vol 1 (commentary), Infotech Ltd, Maidenhead, England.

[6] Bayer, R., and McCreight, E., Organisation and Maintainance of Large Ordered Indexes, Acta Informatica 1, 173-189 (1972)

[7] Challis, M.P., Database Consistency and integrity in a multi-user environment. In: Databases: improving usability and responsiveness. New York: Academic Press, 1978, 245-70.

[8] Codd, E.F., A relational model for large shared databases. Communications of the ACM, 13 (6), June 1970, 377-87.

[9] Cullinane Corporation Integrated Database Management Systems (IDMS) publication: Language programers reference guide.

[10] Date, C.J., An Introduction to the Unified Database Language (UDI), In: Proceedeings of the 1980 conference on Very Large Databases

[11] United States Department of Defence, Reference Manual for the Ada Programming Language, July 1980.

[12] Kaehler, E., Virtual Memory for an Object Oriented Language, Byte August 1981

[13] Lochovsky, F.H., Tsichritizis, D.C., Hierarchical Database Management Systems, ACM Computing Surveys, Vol 8, No 1, March 1978, 105-123

[14] Morrison, R., Davie, A.J.T., Recursive Descent Compiling, Ellis Horwood Ltd Chichester 1981

[15] Morrison, R., S-algol Reference Manual, St Andrews University, 1979

[16] Schmidt, J.W., Some high level language constructs for data of type relation, Transactions on Database Systems, Vol 2, no 3, September 1977 pages 247-281.

[17] Shipman, D.W., The Functional Data Model and the Data Language Daplex, ACM Transactions on Database Systems, Vol 6, No 1, March 1981, 140-73

[18] Smith, J.M., Fox, S., Landers, T., Reference Manual for ADAPLEX, Computer Corporation of America, January 1981

[19] Taylor,R.C. and Frank,R.L., CODASYL database management systems. ACM Computing Surveys,8(i), March 1976, 67-103.

[20] Wasserman, A.I., The data management facilities of PLAIN. Medical Information Science, University of California, San Francisco, 1978.

[21] Whitfield, C., Robertson, P.,The MOUSES reference manual, Moray House College, Edinburgh, 1979.

[22] Wijngaarden, A. van, et al, Revised Report on Algorithmic Language ALGOL 68. Supplement to ALGOL BULLETIN 36, March 1974.

Acknowledgements

We aknowledge the assistance of Richard Marshall in constructing the PS-algol system, the help of Austin Tate and Lee Smith with the presentation of this paper and British Science Research Council for funding our work on grant A865419