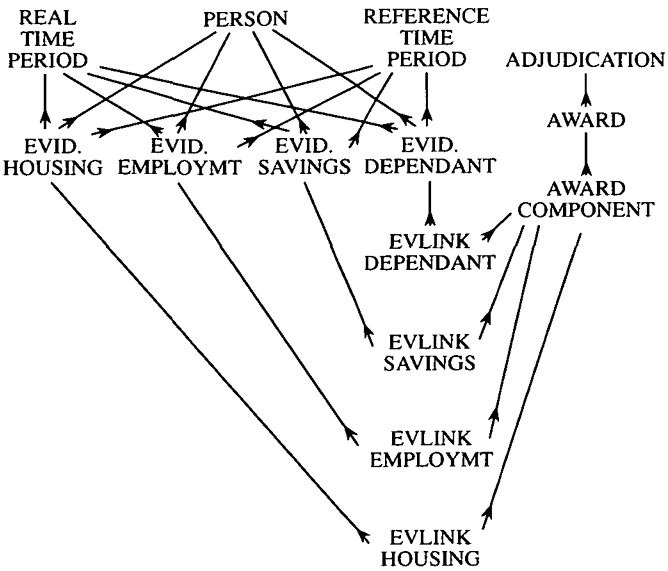

Figure 3. Extended logical data model.

later than the reference-date, or that has an EVID-AMDD-DT that is not 999999, but is later than the reference-date.

5. Conclusions

The design solution outlined in this paper evolved only slowly, after several false starts. Enlightenment occurred with the realisation that the clerical function of 'adjudication' corresponded precisely to 'freezing' all items of a record other than those contained in the standard evidence-header.

Consideration was given to providing REAL-TIME-PERIOD and REFERENCE-TIME-PERIOD as record-types in the physical database structure, to which every evidence record would be physically linked. Our conclusion was, however, that this would lead to unacceptable performance overheads arising from the additional database navigation whenever an evidence record was to be updated. So in the final system, evidence records are chained only to the owning PERSON record, and common routines have been developed that perform the necessary date-matching against candidate records.

The requirement to retain multiple conflicting versions of the 'same' data, as it was believed to be at different points in the past, cannot be unique to the DSS. The design solution described here is now well-proven and is published in the belief that it could be of value to a wider audience.

I. SHEARER

References

1. T. B. Steel, Report on ISO activity in database standardisation. In *Proceedings, International Conference on Databases, Aberdeen, July 1980*, edited S. M. Deen and P. Hammersley. Heyden, London (1980).
2. R. Rock-Evans, *Analysis within the Systems Development Life-Cycle Vol. 1.* Pergamon-Infotech, Maidenhead (1987).
3. S. Jones and P. Mason, Handling the time dimension in a database. In S. M. Deen and P. Hammersley (see 1 above).
4. Gad Ariav, A temporally oriented data model. *ACM Transactions on Database Systems* **11** (4) (1986).

The Reve's Puzzle: An iterative solution produced by transformation

An iterative solution to the Reve's puzzle is produced by largely automatic program transformation from a recursive solution. The result compares favourably with published, manually produced iterative algorithms, both in terms of comprehensibility in its own right, and efficiency.

Received November 1990

1. Introduction

The Reve's Puzzle is an extension of the Towers of Hanoi problem by Dudeney[1] (1907).* The puzzle is posed in terms of 4 stools and a number of cheeses of diminishing size. The rules are identical to those for the Towers of Hanoi.

2. Prologue

Rohl and Gedeon[7] presented a programming solution to the Towers of Hanoi problem with 4 pegs (unaware at the time of Dudeney's book) which used the standard 3 peg solution as a sub-routine as follows:

* Édouard Lucas (1889) mentions a version of Hanoi with 4 or 5 pegs, however, the precise rules are not given. The illustration shows 5 pegs arranged as on a die, with a tower of 16 alternating coloured discs on the central peg, while the text indicates 4 colours for discs *and* pegs. For clarity, it seems best to retain the name *Reve's Puzzle* for the 'standard' 4 peg case, being the earliest unequivocal reference.

```
procedure Hanoi4(n:ndiscs; p1, p2, p3, p4:pegs);
begin
if n > 0 then
  begin
  Hanoi4(n−F(n), p1, p4, p3, p2);
  Hanoi3(F(n), p1, p2, p3);
  Hanoi4(n−F(n), p4, p2, p3, p1)
  end
end
```

The function $F = trunc((sqrt(1+8*n)-1)/2)$ determined how many discs to move using all 4 pegs, and how many with only 3. The function calculated the ordinal number of the triangular number less than or equal to *n*.

This function is clearly not invertible, as it performs a many to one mapping, so the parameter *n* would need to be stacked in an iterative version. Rohl and Gedeon also made the observation that the procedure *Hanoi3* is called with values which decrease by one for each level of the tree of procedure calls of *Hanoi4*, excepting that for a non-triangular number of discs *n* initially, on one level only the same value is used as the previous one. This level varies on *n* in sequence; after all the levels have been 'doubled up' in this fashion, the next triangular number is encountered, after which the cycle starts anew. Table 1 shows the values of *n* for each call to *Hanoi3* from *Hanoi4*.

This is a subtle consequence of the property shown by Roth[9], that the number of moves required for larger towers increases by a number of moves equal to a power of two, a number of times equal to the exponent plus 1. That is, Δ is *1* once, *2* twice, *4* three times, and 2^k '$k+1$' times.

Table 1. The first number indicates the value of *n* in calls to *Hanoi4*, while the list of numbers shows the values of *n* in initial calls to *Hanoi3*.

1	1
2	1, 1
3	2, 1
4	2, 1, 1
5	2, 2, 1
6	3, 2, 1
7	3, 2, 1, 1
8	3, 2, 2, 1
9	3, 3, 2, 1
10	4, 3, 2, 1

We can also see that the number of times for which Δ is $2^{m(n)}$ is repeated is also the same as the depth within the recursive descent on *Hanoi4* that the 'doubling' occurs.

3. The Reve's Puzzle

The above observations lead to the following procedure:

```
procedure (Reve(n:ncheeses);
var position, discontinuity:natural;
  procedure R4(position, skip: natural; s1, s2, s3, s4:stools);
  begin
  if position > 0 then
    begin
    R4(position−1, skip, s1, s4, s3, s2);
    if position > skip then
      Hanoi(position−1, s1, s2, s3)
```

```
    else
      Hanoi(position, s1, s2, s3);
    R4(position − 1, skip, s4, s2, s3, s1)
    end
  end { R4 };
begin { Reve }
position := trunc((sqrt(1 + 8*n) − 1)/2);
discontinuity := n − position*(position + 1)
div 2;
R4(position + 1, discontinuity, 1, 2, 3, 4)
end { Reve };
```

Note that the procedure has also been recast as per Dudeney, using the name *Reve*, and in terms of cheeses and stools. We can therefore revert to using just *Hanoi* for the name of the standard 3 peg problem. A two-level solution has been used to localise initialisations. The variable *position* is now set once, unlike the function *F* which was re-calculated repeatedly.

Analysing the nature of the recursion, we can see that none of the parameters of *R4* need to be stacked.

The variable *position* is modified by the subtraction of *1* at each call, this is invertible, the inverse being *+1*.

Similarly, the stool parameters are only modified in swaps which are their own inverse.

The variable *skip* does not require stacking, as its value is not altered throughout the traversal. Its use as a parameter is good programming practice from a structured viewpoint is not using a global variable, and maintaining a well controlled interface to the rest of the program. The cost of this better structure is a loss of efficiency, due to the allocation and assignment to a local copy at every call to the procedure. This cost is of course recoverable when we eliminate the recursion, which we will do by using the IMP program transformation system by Gedeon.[2]

Briefly, the IMP program transformation system eliminates recursion by the automatic generation of recursive and non-recursive schema pairs. The recursive schema is derived from the recursive procedure by abstraction, while the non-recursive schema is generated from the recursive schema by using an extension of Rohl[8] method of *substitution*, being a symbolic evaluation of the order of execution of the substantive statements of the recursive version.

In this case, the *Reve* procedure performs a complete tree of procedure calls, allowing a specialised non-recursive schema to be generated, similar to those of Partsch and Pepper.[6] A complete tree of procedure calls allows us to consider the symbolic evaluation in terms of traversing the leaves along the bottom of the tree, from left to right rather than climbing up each branch and then back down at each step. This allows us to collect together the recursive ascent and descent simulating code.

The non-recursive result produced for the *Reve* procedure (by IMP) is shown below:

```
begin
if odd(position + 1) then
  swap(s4, s2);
for S := 1 to Power2(position + 1) − 1 do
  begin
  Decompose2(S, level);
  if odd(level) then
    swap(s1, s4);
  swap(s2, s4);
  if 1 + level > skip then
    Hanoi(level, s1, s2, s3)
  else
    Hanoi(1 + level, s1, s2, s3);
  swap(s4, s1);
  if odd(level) then
    swap(s4, s2)
  end
end
```

The **for** loop on *S* traverses the leaves, the procedure *Decompose2* discovers how far up the tree the next substantive statement is, and which recursive call it is 'in'. The parameter *position* controls the recursive descent and is largely subsumed by the schema variable *level*, which is extracted from *S* as needed.

This can be simplified further within IMP, with user guidance of the sequence and application of transformations, which are verified for correctness and applicability by the system.

We will now skip directly to the simplified version, which is now as follows:

```
procedure Reve(n : n cheeses);
type natural = 0..maxint;
var position, discontinuity: natural;
  s1, s2, s4: nstools;
  S, level, nap: integer;
begin
position := trunc((sqrt(1 + 8*n) − 1)/2);
discontinuity := n − position*(position + 1)
div 2;
s1 := 1;
s2 := 2;
s4 := 4;
if odd(position + 1) then
  swap(s4, s2);
for S := 1 to Power2(position + 1) − 1 do
  begin
  Decompose2(S, level);
  if 1 + level > discontinuity then
    nap := level
  else
    nap := 1 + level;
  if odd(level) then
    Hanoi(nap, s4, s1, 3)
  else
    Hanoi(nap, s1, s4, 3);
  rotate3(s1, s2, s4)
  end
end { Reve };
```

Note that the procedure *R4* is no longer necessary being called only once. The variables *position* and *discontinuity* used as actual parameters are not otherwise used in *Reve* and are substituted for their formal parameters in the statements. Similarly, the stool parameters are declared in *Reve*, except for *s3*, the use of which does not vary, and is automatically replaced by the actual parameter expression *3*.

The solution still involves recursion, within the calls to *Hanoi*, but only until we follow a simpler and similar path of transformation and also convert it to iterative form.

4. Conclusions

We have produced an iterative algorithm for the Reve's puzzle which compares well with previous versions. The algorithm is certainly easier to understand, and has much lower space requirements than Lu,[4] and also has an efficiency lead on that of the quite similar algorithm by Hinz[3] which requires the use of exponentiation by powers other than 2, which can not therefore be implemented by efficient shift instructions on compilation.

Most importantly, we have produced the above program automatically, the interactive manipulations of the program are largely cosmetic, the bulk of the work had already been done by the IMP program transformation system.

Finally, in this process we have also shown by transformation some interesting properties of the Reve's puzzle, properties which others have assumed.

This iterative algorithm for the Reve's puzzle is similar to the well known iterative solution to the Towers of Hanoi problem, the stools are arranged in a circle, and the towers in alternate levels move in opposite directions. It is surprising from a purely intuitive viewpoint that the rotation is again of length 3 not 4. Also interesting is that stool 3 is always the spare stool when moving discs using 3 stools; in retrospect this is also clear from the original recursive version.

Acknowledgements

I would like to acknowledge the guidance and encouragement of Jeff Rohl.

GEDEON, T. D.

Department of Computer Science, Brunel University, Uxbridge, Middlesex, UB8 3PH.

References

1. H. E. Dudenay, *The Canterbury Puzzles*, (4th Edition of 1919 reprinted, and published by Dover 1958), Thomas Nelson & Son, London (1907).
2. T. D. Gedeon, *IMP: An Interactive Program Transformation System*, PhD Thesis, The University of Western Australia (1989).
3. A. M. Hinz, An iterative algorithm for the tower of Hanoi with four pegs, *Computing* **42**, 133–140 (1989).
4. X.-M. Lu, An iterative solution for the 4-peg Towers of Hanoi, *The Computer Journal*, **32**, (2), 187–188 (1989).
5. E. Lucas, Nouveaux Jeux Scientifiques, *La Nature* **17**, 301–303 (1889).
6. H. Partsch and P. Pepper, A Family of Rules for Recursion Removal, *Inf Proc Letters*, **5**, (6), 174–177 (1976).
7. J. S. Rohl and T. D. Gedeon, Four Tower Hanoi and beyond, *Proceedings of 6th Australian Computer Science Conference*, 156–162, Sydney (1983).
8. J. S. Rohl, *Recursion via Pascal*, Cambridge University Press, Cambridge (1984).
9. T. Roth, The Tower of Brahma Revisited, *J. Recreational Mathematics*, **7**, (2), 116–119 (1974).