ASTRE: Towards A Fully Automated Program Transformation System

F. Bellegarde*
Pacific Software Research Center
Oregon Graduate Institute of Science & Technology

1 Introduction

It has often been said that functional programs are constructed using functions as pieces. Data structures such as lists and trees are the glue to hold them together. This compositional style of programming produces many intermediate data structures. One way to circumvent this problem is to perform *fusion or deforestation* on programs. Deforestation algorithms (elimination of useless intermediate data-structures) [6, 10] do not recognize that an expression contains two or more functions that consume the same data structure. These functions can be put together in a tuple as a single function that traverses the data structure only once. This tactical is usually called two-loops fusion or two-loops tupling since it is implemented by using a tupling technique. It has been pointed out by Dershowitz [8] that an fold-unfold methodology [5] can be controlled by a completion procedure. Following this idea, the transformation system Astre [2] is based on completion procedures.

2 The Astre System

The transformational approach to the development of programs is attractive for writing small components of large software systems. This approach, to be effective, must be fully automated so that it is not necessary to be an expert in transformation strategies to use the transformational approach for software design. A prototype of a fully automated mode of Astre is a component of the tool suite that support a Method for Software Design for Reliability and Reuse developed in the Pacific Software Research Center [1]. The tool suite provides a translation of ML programs into a rewrite system input of Astre. It includes an implementation of the Chin and Darlington's specialization algorithm [7]. for conversion to first-order. The result is always an orthogonal (left-linear and non-overlapping) constructor-based (a constructor-based system of equalities is similar to set of definition equalities with pattern-matching arguments in functional programming) rewrite system R_0. In Astre, synthesis by completion is used as a

*The author was supported by a contract with Air Force Materiel Command (F19628-93-C-0069).

mechanism to transform R_0 into a sequence of orthogonal, terminating and constructor based rewrite systems $R_1, R_2, \ldots, R_n$ to get a new, semantically equivalent ML program P_n which is more efficient. Astre translates R_n into a ML program where functions are presented by a set of mutually recursive functions with pattern-matching arguments. A fully automatic version of Astre automatizes deforestation and two-loops fusion strategies. A semi-automatic mode authorizes the user to input laws to facilitate a deforestation. Number of issues occur to automate the synthesis process.

- Generation of useless critical pairs is the major drawback for using completion in its application to synthesis. Astre arefully controls the production of critical pairs hence ensuring termination of the completion [4].
- Astre controls the orientation of the critical pairs into rules as required by the transformation strategy. It guarantees that termination is preserved for a constructor-based orthogonal rewrite system [4].
- Furthermore, given a tactical for transformation, Astre ensures the termination of the sequence of syntheses from the source rewrite system R_0 into the succession of synthesized rewrite systems $R_1, R_2, \ldots$ [4].
- Synthesis rules introduce a new function to synthesize. In the fold-unfold methodology synthesis rules are called *definition rules* or *eurekas rules* because they are introduced through the insight of a clever user. Mechanisms to generate automatically a synthesis rule for deforestation and two-loops fusion strategies are presently implemented in Astre.
- Moreover a set rewrite rules (inductive theorems of R) can be input to the synthesis process in the semi-automatic mode. These *laws* usually facilitate the process in the fold-unfold method.

However, we have noticed that most of the situations that requires laws in a deforestation can be simply handled by introducing additional synthesis rules like $length(append(x, y)) \rightarrow h(x, length(y))$. Such a synthesis rule helps the symbol *length* to go down to consume a term that substitutes the variable y as well as the inductive law $length(append(x, y)) = append(length(x), length(y))$ can do. This technique does not work for pushing *length* to go down towards the inductive variable x. If this is needed the deforestation fails. Let us compare the results given by the fully automatic version of Astre and the semi-automatic mode on the following example.

3 Examples

The pencil and paper transformation of the functional program is presented by S. Thompson [9]. The problem solved by this program is stated as follows by S. Thompson:

> Given a finite list of numbers, find the maximum value for the sum of a (contiguous) sublist of the list.

Numbers can be positive as well as negative integers. Let us begin with the quadratic in the length of the list first-order ML program in Figure 1. where *I_plus*, and *I_max* are library

$$
\begin{array}{lll}
\textbf{fun}\ \mathit{My\_append}\ \mathit{My\_nil}\ x & = & x \\
\quad |\ \mathit{My\_append}\ (C(x, xs))\ y & = & (C(x, \mathit{My\_append}\ xs\ y)); \\
\textbf{fun}\ \mathit{map\_cons}\ x\ \mathit{My\_nil} & = & \mathit{My\_nil} \\
\quad |\ \mathit{map\_cons}\ x\ (C(y, ys)) & = & (C(C(x, y), \mathit{map\_cons}\ x\ ys)); \\
\textbf{fun}\ \mathit{frontlists}\ \mathit{My\_nil} & = & C(\mathit{My\_nil}, \mathit{My\_nil}) \\
\quad |\ \mathit{frontlists}\ (C(x, xs)) & = & \mathit{My\_append}\ (\mathit{map\_cons}\ x\ (\mathit{frontlists}\ xs)) \\
 & & (C(\mathit{My\_nil}, \mathit{My\_nil})); \\
\textbf{fun}\ \mathit{sublists}\ \mathit{My\_nil} & = & C(\mathit{My\_nil}, \mathit{My\_nil}) \\
\quad |\ \mathit{sublists}\ (C(x, xs)) & = & \mathit{My\_append}\ (\mathit{map\_cons}\ x(\mathit{frontlists}\ xs))\ (\mathit{sublists}\ xs); \\
\textbf{fun}\ \mathit{sum}\ \mathit{My\_nil} & = & 0 \\
\quad |\ \mathit{sum}\ (C(x, xs)) & = & \mathit{I\_plus}\ x\ (\mathit{sumxs}); \\
\textbf{fun}\ \mathit{map\_sum}\ \mathit{My\_nil} & = & \mathit{My\_nil} \\
\quad |\ \mathit{map\_sum}\ (C(x, xs)) & = & (C(\mathit{sum}\ x, \mathit{map\_sum}\ xs)); \\
\textbf{fun}\ \mathit{fold\_max}\ (C(x, \mathit{My\_nil})) & = & x \\
\quad |\ \mathit{fold\_max}\ (C(x, C(y, z))) & = & \mathit{I\_max}\ x\ (\mathit{fold\_max}\ (C(y, z))); \\
\textbf{fun}\ \mathit{maxsub}(x) & = & \mathit{fold\_max}(\mathit{map\_sum}(\mathit{sublists}(x)));
\end{array}
$$

Figure 1: Source Program

functions. The automatic mode of Astre yields the result presented in Figure 2 in $11.29s$. Astre performs nine syntheses from which seven successful deforestations, one additional synthesis to help deforestation, and one two-loops fusion. One deforestations fails. It corresponds to an attempt to eliminate the intermediary list produced by *A_sym3* and consumed by *A_sym2_1* in the definition of *A_sym5*. Astre does not consider for deforestation an intermediary list produced by a recursive call like *A_sym3* and consumed by *A_sym1*. Such situations are more relevant to a derecursion tactical and are the sources of failure in the absence of laws. Except these two compositions, the program in Figure 2 is completely deforested. In other words, there is no other intermediary lists produced by a function and consumed by another one. Run-time difference between the source and the transformed program are not significative. Recall that the goal of a deforestation in not to improve the run-time but only elimination of useless intermediate data structures without loss of run-time efficiency. However, if, using the semi-automatic mode, the user cleverly provides the following inductive laws at the level of the second synthesis:

$$
\begin{array}{l}
\mathit{map\_sum}\ (\mathit{A\_sym1}\ x\ y\ z) = \mathit{My\_append}\ (\mathit{map\_plus}\ x\ (\mathit{map\_sum}\ y))\ (\mathit{map\_sum}\ z) \\
\mathit{fold\_max}\ (\mathit{My\_append}\ x\ y) = \mathit{I\_max}\ (\mathit{fold\_max}\ x)\ (\mathit{fold\_max}\ y) \\
\mathit{fold\_max}\ (\mathit{map\_plus}\ x\ y) = \mathit{I\_plus}\ x\ (\mathit{fold\_max}\ y)
\end{array}
$$

```
fun A_sym1 x1 (C(x2,x3)) x4      =  C((C(x1,x2)),(A_sym1 x1 x3 x4))
  | A_sym1 x2 My_nil x1          =  x1;
fun sum My_nil                   =  0
  | sum (C(x1,x2))               =  I_plus x1 (sumx2);
fun A_sym3 x1 My_nil x2          =  C((C(x1,My_nil)),x2)
  | A_sym3 x1 (C(x2,x3)) x4      =  A_sym1 x1 (A_sym3 x2 x3
                                    (C(My_nil,My_nil))) x4;
fun A_sym6 x1 (C(x2,x3))         =  I_max x1 (A_sym6 (sum x2) x3)
  | A_sym6 x1 My_nil             =  x1;
fun A_sym4 x1 x2 (C(x3,x4)) x5   =  I_max x1
                                    (A_sym4 (I_plus x2 (sum x3)) x2 x4 x5)
  | A_sym4 x1 x2 My_nil x3       =  A_sym6 x1 x3;
fun A_sym2_1 x1 (C(x2,x3)) x4 x5 =  A_sym4 (I_plus x1 (sum x2)) x1 x3 x4
  | A_sym2_1 x1 My_nil x2 x3     =  x3;
fun A_sym5 x1 My_nil x2 x3       =  A_sym6 (I_plus x1 0) x2
  | A_sym5 x1 (C(x2,x3)) x4 x5   =  A_sym2_1 x1
                                    (A_sym3 x2 x3 (C(My_nil,My_nil))) x4 x5;
fun A_sym8 My_nil                =  ((C(My_nil,My_nil)),0)
  | A_sym8 (C(x1,x2))            =  let (u,v) = A_sym8 x2 in ((A_sym3 x1 x2 u),
                                    (A_sym5 x1 x2 u v)) end;
fun maxsub x1                    =  second (A_sym8 x1);
```

Figure 2: Fully Automatic Output

where *map_plus* is defined by:

$$map\_plus\ x\ My\_nil = My\_nil$$
$$map\_plus\ x\ (C(y, ys)) = My\_append\ (I\_plus\ x\ y)\ (map_p lus\ ys)$$

Astre outputs:

```
fun A_sym4 My_nil          =  (0,0)
  | A_sym4 (C(x1,x2))      =  let (u,v) = A_sym4 x2 in
                              ((I_max (I_plus x1 u) 0),
                               (I_max (I_plus x1 u) v)) end;
fun maxsub x1              =  second (A_sym4 x1);
```

after three deforestations and one two-loops fusion. No deforestation fails. Moreover, the laws allow to improve the complexity. The program is linear in the length of the list. Thompson's

pencil and paper's transformation gives the same result modulo the two-loops fusion.

$$\begin{array}{lll} \textbf{fun}\ \mathit{maxfrontMy\_nil} & = & 0 \\ \quad \mid \mathit{maxfront}\ (C(x,y)) & = & \mathit{bimax}\ 0\ (\mathit{I\_plus}\ x\ (\mathit{maxfront}\ y)); \\ \textbf{fun}\ \mathit{maxsubMy\_nil} & = & 0 \\ \quad \mid \mathit{maxsub}\ (C(x,y)) & = & \max\ (\mathit{maxsub}\ y)\ (\mathit{I\_plus}\ x\ (\mathit{maxfront}\ y)); \end{array}$$

4 Conclusion

Astre has achieved its initial goal: it is fully automatic for deforestation and two-loops fusion of functional programs. Other fully automatic algorithms for deforestation does not include two-loops fusion. They reject all deforestations that necessitate laws but Astre can perform most of them using additional synthesis. Moreover they do not extend easily to include laws or other strategies. The limitation of Astre is the termination obligation of the input rewrite system. Also Astre does not process easily a large amount of rules. The present prototype has been used so far up-to 500 rules input. At this size, it becomes intractable to perform all the syntheses. We plan to automatize derecursion tactical and automatic insertion of laws [3] in a near future.

References

[1] J. Bell et al. Software Design for Reliability and Reuse: A proof-of-concept demonstration. Technical report, Department of Computer Science and Engineering, Oregon Graduate Institute, 1994. To be presented at the Tri-Ada conference in November.

[2] F. Bellegarde. Program Transformation and Rewriting. In *Proceedings of the fourth conference on Rewriting Techniques and Applications*, volume 488 of *LNCS*, pages 226–239. Springer-Verlag, 1991.

[3] F. Bellegarde. A transformation system combining partial evaluation with term rewriting. In *Higher Order Algebra, Logic and Term Rewriting (HOA '93)*, volume 816 of *LNCS*, pages 40–58. Springer-Verlag, September 1993.

[4] F. Bellegarde. Termination issues in automated syntheses. Submitted to RTA95, September 1994.

[5] R. M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24:44–67, 1977.

[6] W. Chin. Safe Fusion of Functional Expressions II: Further Improvements. *Journal of Functional Programming*, 11:1–40, 1994.

[7] W. Chin and J. Darlington. Higher-Order Removal: A modular approach. Unpublished work, 1993.

[8] N. Dershowitz. Completion and its Applications. In *Resolution of Equations in Algebraic Structures*. Academic Press, New York, 1988.

[9] S. Thompson. *Type Theory and Functional Programming*. Addison Wesley, 1991.

[10] P. Wadler. Deforestation: Transforming Programs to eliminate trees. In *Proceedings of the second European Symposium on Programming ESOP'88*, volume 300 of *LNCS*. Springer-Verlag, 1988.