Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

279

Joseph H. Fasel Robert M. Keller (Eds.)

Graph Reduction

Proceedings of a Workshop
Santa Fé, New Mexico, USA
September 29–October 1, 1986

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

Editors

Joseph H. Fasel
University of California
Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA

Robert M. Keller
Quintus Computer Systems, Inc.
1310 Villa Street
Mountain View, California 94041, USA

This volume was edited under the auspices of the U.S. Department of Energy.

CR Subject Classification (1987): F.1.1, F.4.1, C.1, D.1.1, D.3.4

ISBN 3-540-18420-1 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-18420-1 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210

Preface

On September 29 through October 1, 1986, a workshop on graph reduction was held in Santa Fe, New Mexico, USA, sponsored by the Los Alamos National Laboratory of the University of California and Microelectronics and Computer Technology Corporation (MCC). 66 researchers from the United States, Canada, the United Kingdom, Sweden, Finland, the Netherlands, France, West Germany, and Australia participated in the workshop, which consisted of 28 presentations, combined with much informal discussion. The papers in the present volume are based on the presentations, and because the final versions were prepared after the workshop, they reflect some of the discussions, as well.

Historical Comment

The term *graph reduction,* referring to an evaluation model for functional languages, was probably first used in the dissertation of Christopher Wadsworth (Oxford, 1971). Similar ideas had earlier been discussed in a thesis at M.I.T. (Suhas S. Patil, M.S. Thesis, *An abstract parallel processing system,* 1967). That numerous investigations of this concept have sprung forth attests to the concept's elegance and seductiveness. Witness the present collection of papers, which represents contributions from a sizeable, but incomplete, subset of the workers in this community.

Rationale

Some of the benefits of graph reduction can be found in the following traits:

- A mathematically elegant denotational semantics
- Lazy evaluation, which avoids recomputation and makes programming with infinite data structures (such as streams) possible.
- A natural tasking model for fine-to-medium grain parallelism (see *Keller, Lindstrom, and Patil, A loosely-coupled applicative multiprocessing system, Proc. NCC 1979).*

Brief Introduction to Graph Reduction

The concept of graph reduction can be illustrated by considering a functional expression, such as

$$((a + b) / (c * d)) - ((c * d) / (e + f)),$$

the graphical equivalent of which is shown in Figure 1.

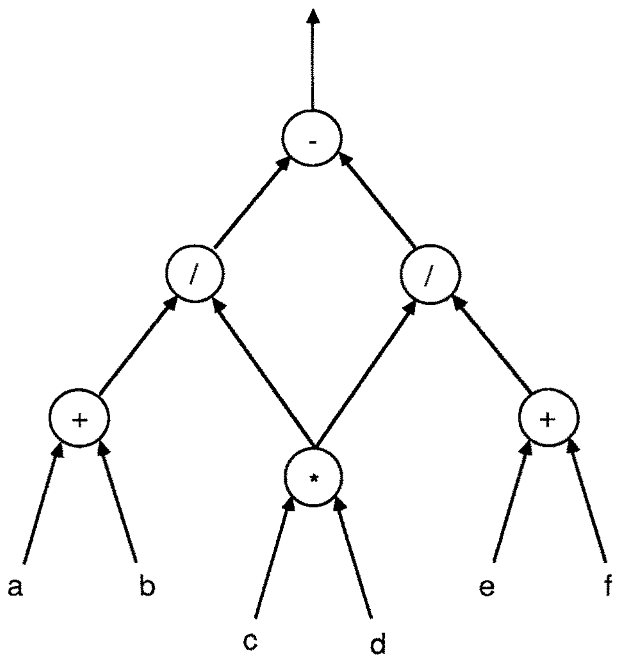


Figure 1

In the figure, the leaves *a, b, c, d, e, f* are assumed to represent values. This graph can be *reduced* by selecting any evaluable operator node; here, the evaluable nodes are those whose inputs are leaves. There are three such operators in the Figure. *Reduction* of such a node is represented by replacing the node with its value, representable symbolically as *(a + b)*, *(c * d)*, and *(e + f)* for each of the three choices in the Figure. Figure 2 shows the result of each of three possible reductions. In practice, only one reduction need be performed at a time, but the *independence* of the three reductions is indicative of the potential *parallelism* in the reduction process. Note also that the subgraph for *(c * d)* is shared, and thus need only be reduced once. This is an advantage of graph reduction, as opposed to string or tree reduction.

Figure 3 begins by showing the result of applying all three reductions, and then continues by applying other reductions, until the result is a single value, again represented symbolically.

In a practical system, we cannot be content with evaluating fixed graphs; we need to be able to build parameterized graphs and reduce them. For example, Figure 4 shows a graph representing the product of the integers from *m* through *n*.

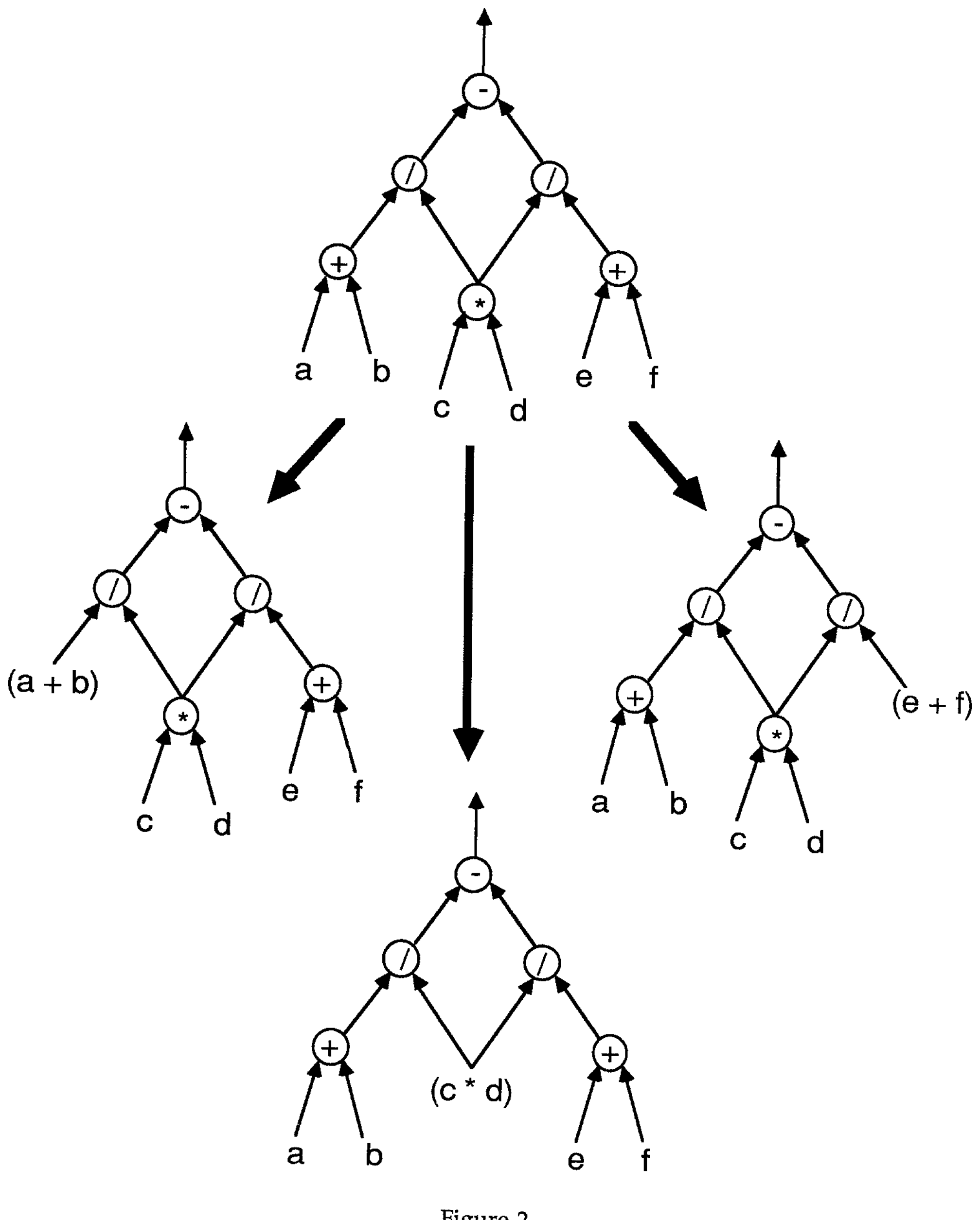


Figure 2

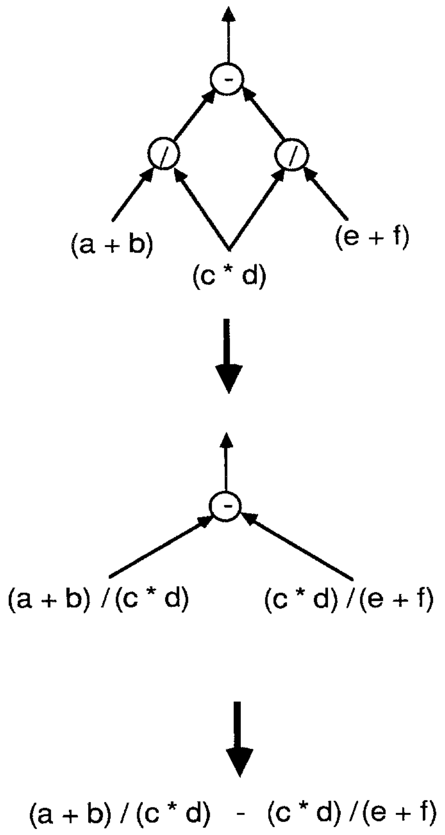


Figure 3

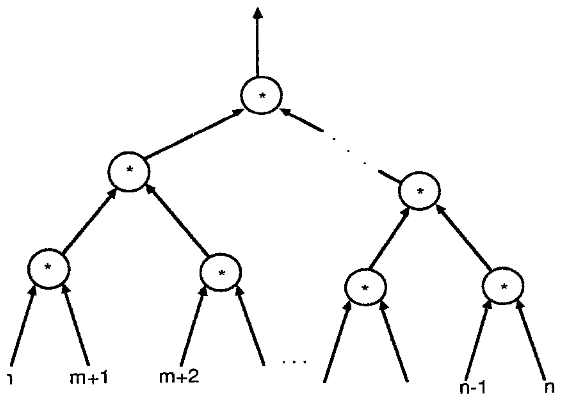


Figure 4

Suppose we wish to construct such a graph for given *m* and *n* and then reduce it. This may be accomplished with a *graph grammar production,* as shown in Figure 5. The idea here is that a node labelled *product* is to be *expanded* into a graph before any reduction. Again, the computation is viewed as a succession of transformations from one graph to another. A production can be applied anywhere in the graph that there is a node corresponding to its left-hand side. The application of this production entails replacing that node with the corresponding right-hand side, splicing in such a way that argument correspondence is preserved. As with reduction, expansion can take place in parallel at several sites within the graph, since the graph grammar is essentially "context-free".

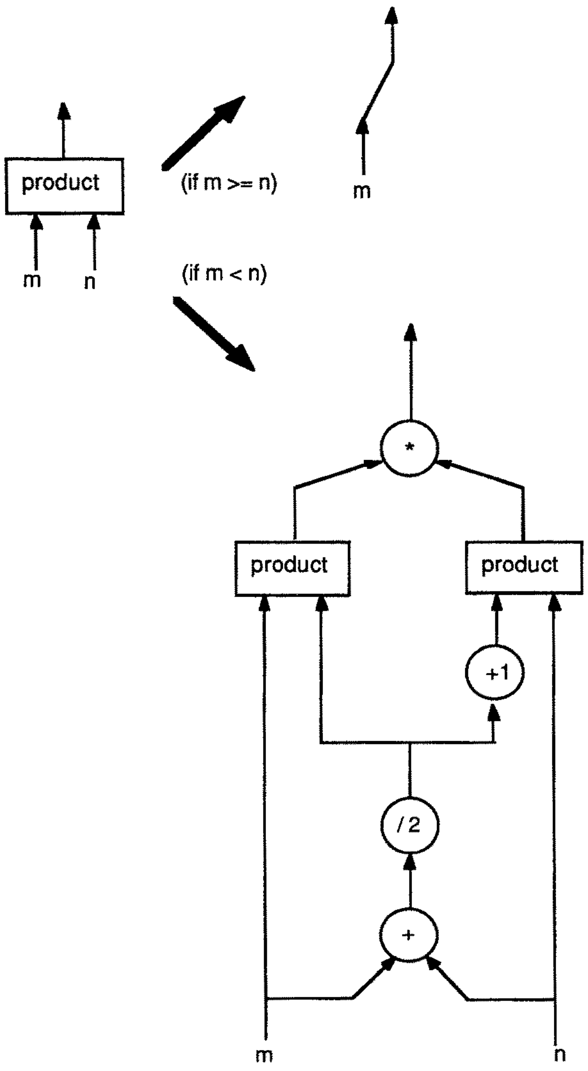


Figure 5

We show in Figure 5 two expansions, conditioned on the relation between *m* and *n*, that must be known before the corresponding right-hand side can be chosen. However, we can achieve the same effect with a single production by introducing a conditional (*if_then_else_*) node, as shown in Figure 6.

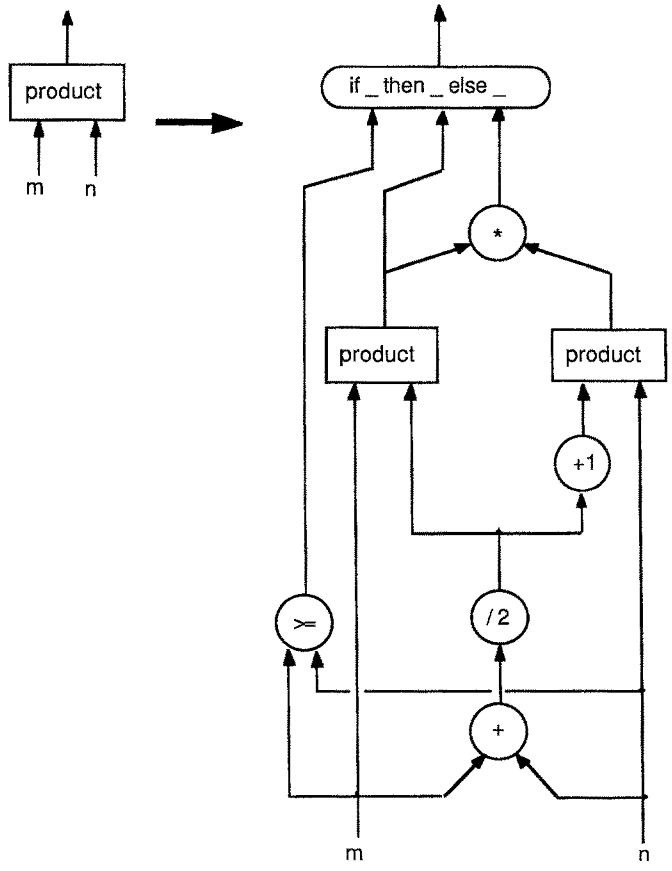


Figure 6

We assume that the conditional node is reduced *lazily*; that is, the *then* and *else* parts are not evaluated before the node is reduced. The principle of *lazy* (or *demand-driven*) evaluation is that no reduction or expansion is performed until its value is needed. This evaluation scheme has the benefit of avoiding redundant computation, and in particular, allows us to use infinite data structures, so long as only finite portions of them are ever demanded. For example, Figure 7 shows an expansion rule for the infinite list (or *stream*) of successive integers beginning with *n*.

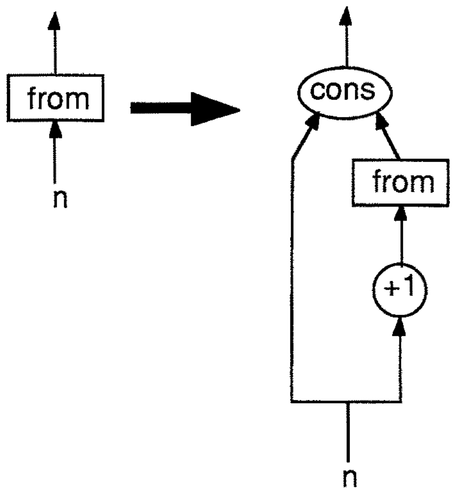


Figure 7

Figure 8 shows the configuration of the stream of integers from 0 after the first two elements have been demanded.

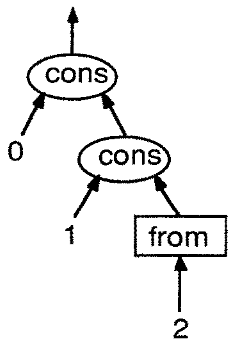


Figure 8

There is, of course, a connection between graph expansion and *beta-reduction* in the lambda calculus. The production itself corresponds to a lambda expression, and the position of the expanded node within the graph corresponds to its arguments, as in the following indication of function application:

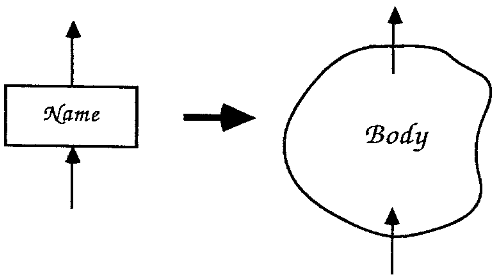
(λ(*formals*).*body*) *actuals* --> *body*(with actuals substituted for formals)

Lazy evaluation of graphs corresponds to *normal-order reduction* in the lambda calculus, but need not suffer from the inefficiency associated with the latter, because multiple occurrences of a formal parameter in a lambda abstraction, which result in reevaluation of the corresponding actual parameter, can be represented in the equivalent graph by references to a shared subgraph (the actual), which will be evaluated at most once. It is also noteworthy that the graph representation eliminates the need for the traditional *alpha-reduction*, as the arcs corresponding to bound variable instances in the lambda calculus are anonymous in the graph calculus.

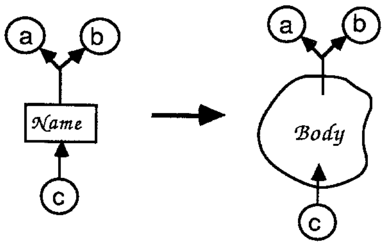
The connection between graph reduction and lambda calculus needs to be completed by mentioning briefly the correspondence to *free variables* in lambda expressions. These are variables that are expected not to get their values from arguments of the immediate lambda binding, but rather to inherit them from outside. This correspondence is best demonstrated by an alternate representation of the production, using an *encapsulated body*. Instead of a named node, which would have been expanded, we use an *apply* node, which expects an encapsulated body as one of its arguments, and which is defined to have a reduction rule resulting in the substitution of the body for the apply node. Figure 9 shows the original production in part (a), the result of beta reduction in part (b), and the equivalent reduction using an *apply* node in part (c).

Now we can use encapsulated bodies to show the equivalent of free variables. These are bodies with arc values entering from *outside* the node, as shown in Figure 10. These values are bound before expansion of the *apply* node, not during. Using this representation, techniques of the lambda calculus can be explained (see *R.M. Keller, Semantics and applications of function graphs, University of Utah 1980*).

(a)



(b)



(c)

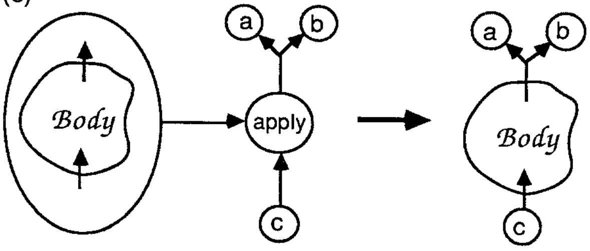


Figure 9

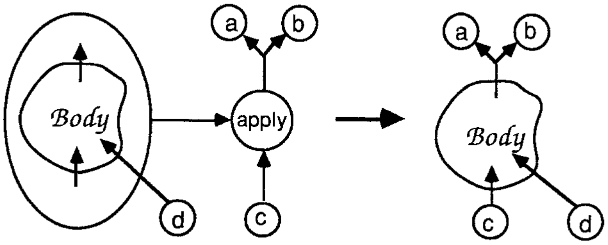


Figure 10

Open Issues

It is scientifically worthwhile to examine issues such as

- whether graph reduction is the most efficient way to achieve the benefits mentioned in the opening paragraphs (*e.g.* can variants of the SECD model do just as well, or should these variants properly be viewed as instances of graph reduction)

- whether some of the benefits (*e.g.* natural tasking granularity) apply just as well to models that are not purely functional

- whether graph reduction can take on a wider meaning than as an evaluation model for the lambda-calculus, (*e.g.* in a computational symbolic algebra, where the data are *essentially* graphs)

- whether the graph reduction model limits the scope of programming model that can be used (*e.g.* can naturally represent *continuation-based* programming, on which a wide variety of control structures can be represented functionally)

Contributed Papers

One can find in the papers contributed to the workshop concern for some of the issues mentioned above and related ones. We have grouped the papers into categories, as shown in the Table of Contents. In an area where there is significant interplay of concerns (efficiency, parallelism, language capability, *etc.)*, no such categorization can be perfect. Thus, we have merely attempted a coarse grouping to aid ourselves in introducing the topics of concern in each of the papers.

In the category of models for graph reduction, we find papers exposing various views of graph reduction as a means for language implementation. Hans-Georg Oberhauser (*On the correspondence of lambda style reduction and combinator style reduction*) discusses the relationship between two styles of graph reduction, one based on beta reduction in the lambda calculus and the other based on a combinator calculus. Further connections along these lines are exposed by Klaus Berkling (*Head order reduction: A graph reduction scheme for the operational lambda calculus*) and Jon Fairbairn (*A simple abstract machine to execute supercombinators*). Joseph Goguen, Claude Kirchner, and José Meseguer (*Concurrent term rewriting as a model of computation*) discuss the techniques and motivation underlying their Rewrite-Rule Machine project.

An important set of issues surrounds the problems of implementing graph reduction on existing architectures. Benjamin Goldberg and Paul Hudak (*Alfalfa: Distributed graph reduction on a hypercube multiprocessor*) present their implementation work for a multiprocessor with distributed processors and memory. Randy Michelson, Lauren Smith, Elizabeth Williams, and Bonnie Yantis (*Parallel Graph Reduction on a Supercomputer: A Status Report*) present related work oriented toward existing supercomputer implementations. Thomas Johnsson (*Target code generation from G-machine code*) describes a method of functional language compilation using a graph-reduction model.

The graph reduction concept has inspired several architectures specially designed to exploit the parallelism it provides. The papers here reflect work as it has proceeded in three different countries toward such ends: Michel Castan, Guy Durrieu, Bernard Lecussan, Michel Lemaître, Allesandro Contessa, Eric Cousin, and Paulino Ng (*Toward the design of a parallel graph reduction machine: The MaRS project*), P.G. Harrison and M.J. Reeve (*The parallel graph reduction machine, Alice*), and Robert M. Keller, Jon W. Slater, and Kevin T. Likes (*Overview of Rediflow II Development*).

Along with such architectures come issues of how best to allocate physical resources, such as processors and memory. Included in this category are papers by M.C.J.D. van Eekelen and M.J. Plasmeijer (*Specification of reduction strategies in term rewriting systems*), F. Warren Burton (*Controlling reduction partial order in functional parallel programs)*, Ashoke Deb (Parallel garbage collection for graph machines), and Ian Watson and Paul Watson (*Graph reduction in a parallel virtual memory environment*).

For proposed architectures, it is important to assess performance before constructing hardware. The papers by Richard B. Kieburtz (*Performance measurement of a G-machine implementation*) and Steven Tighe, Ken Zink, Richard Brice, and William Alexander (*A flexible architectural study methodology*) present results and approaches in this area.

One topic that generated particularly lively discussion at the workshop was that of arrays in functional languages and their implementation by graph reduction. This discussion was inspired by presentations by Paul Hudak on incremental and monolithic array operations and by Arvind on I-structures. This discussion motivated Philip Wadler to devise a new monolithic array operation and to write a paper about it, which he distributed to an electronic mailing list on functional programming. Some further discussion on this subject then followed on the mailing list, and the papers by Paul Hudak (*Arrays, non-determinism, side-effects, and parallelism: A functional perspective*), Philip Wadler (*A new array operation*), and Arvind, Rishiyur S. Nikhil, and Keshav K. Pingali (*I-structures: Data structures for parallel computing*) bear some of the fruit of this discussion.

The final category of papers concerns another area of intense activity with regard to language concepts implementable by graph reduction, that of *logic programming*. These are the papers by Bharat Jayaraman and Gopal Gupta (*Parallel execution of an equational language*), Gary Lindstrom (*Implementing logical variables on a graph reduction architecture*), Uday S. Reddy (*Functional logic languages, Part I*), and John Staples and Peter J. Robinson (*Unification of quantified terms*).

We hope the reader will find these papers useful both as informative sources of current work in the area of graph reduction and as points of departure for future work.

Acknowledgements

The workshop was originally suggested by Alfred Hartmann of MCC, and was to be sponsored by Los Alamos, MCC, and the Burroughs Austin Research Center. Unfortunately, not long after planning for the workshop began, BARC was closed.

In addition to the committee members listed below, three others participated in the early phases of organizing the workshop: Henry Tirri, of the University of Helsinki and MCC, and Carl Pixley and Hamilton Richards, then with BARC and now with MCC and the University of Texas at Austin, respectively.

The committee are grateful for assistance and encouragement from several individuals and organizations in Europe. In particular, we thank Ronan Sleep, University of East Anglia; Simon Peyton Jones, University College London; the Alvey Directorate, London, Brian Oakley, Head; and the Division of Software Technology and Advanced Information Processing, Commission of the European Communities, Brussels, John Elmore, Head.

The editors are grateful to Patricia Fasel, Los Alamos National Laboratory, for her generous assistance in the editing and preparation of this volume.

Los Alamos's contribution to the funding of the workshop was provided by the U.S. Department of Energy, Office of Energy Research.

Robert M. Keller
Mountain View, California

Joseph H. Fasel
Los Alamos, New Mexico

June, 1987

Table of Contents