Chapter 5

Architecture, Compilers and Parallel Computation
Compiling Concurrent Rewriting onto the Rewrite Rule Machine*

Hitoshi Aida†, Joseph Goguen‡ and José Meseguer
SRI International, Menlo Park CA 94025

Abstract
Following a brief review of the of the Rewrite Rule Machine's concurrent rewriting model of computation, this paper describes a technique for transforming rule sets which eliminates non-left linear and conditional rules. By Theorem 1, this transformation preserves termination and result; it is used as a first phase of compilation. After a brief review of RRM architecture, the second phase of compilation is described, which transforms unconditional left linear rule sets into ensemble micro code. By Theorem 2, this transformation preserves the time complexity of a set of rewrite rules executed in parallel. The paper concludes with a discussion of the ensemble controller's role.

1 Introduction
The Rewrite Rule Machine (RRM) Project resembles many other projects in its goal of building a high performance parallel graph reduction machine for declarative languages. However, it differs significantly from other projects in its choice of architecture and languages.

The RRM architecture is a multi-level hierarchy that encompasses both fine-grain SIMD and coarse-grain MIMD computation, and eliminates the von Neumann bottleneck between processor and memory by processing data where it is stored. The lowest level of the RRM is the cell, which combines processing, storage, and communication capabilities. Many cells and a controller are laid out on a chip called an ensemble, which operates in SIMD mode. The total RRM is a network of ensembles operating in MIMD mode.

The RRM model of computation, called concurrent rewriting, is designed to mediate between this architecture and various programming languages. The software side of the RRM project has been exploring declarative multi-paradigm languages which support various combinations of functional, object oriented, and logic programming.

This paper presents some techniques, embodied in a prototype compiler, for compiling equations into SIMD ensemble code, which is then distributed across many ensembles to perform a computation in a globally MIMD mode. Since the techniques are developed for the abstract model of computation, they should be applicable to compiling other languages onto other parallel machines.

The three major stages of the compilation process are as follows:

1. Translate source code into a set of (conditional) rewrite rules. The relatively straightforward translation techniques for functional languages extend to object-oriented and (relational) logic programming and various combinations [18].

2. Eliminate conditions and left-non-linearities from the rules. This is very important for SIMD parallel execution, since finding a match for an (unconditional) left-linear rule only requires local structural checks that can be easily achieved by broadcasting a sequence of SIMD instructions, but this is not at all the case for more general rules; see Section 3.

*Supported by Office of Naval Research Contracts N00014-86-C-0450 and N00014-90-C-0210, and NSF Grants CCR-8707155 and CCR-9007010.

†Department of Electrical Engineering, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113 Japan; work done at SRI International.

‡Oxford University, Programming Research Group, Oxford OX1 3QD, United Kingdom.