1995

Parallelization of Reconstructability Analysis Algorithms.

Patti E. Iles
Louisiana State University and Agricultural & Mechanical College

PARALLELIZATION OF RECONSTRUCTABILITY ANALYSIS ALGORITHMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Patti E. Iles
B.S., Louisiana State University, 1989
May 1995

Dedication

This work is dedicated to the memory of my mother, Eddie Huffman Iles, who always expected the best in me and in doing so always motivated me to be my best.

Mom, I did it!

Acknowledgements

First, I would like to express my appreciation to Dr. J. Bush Jones. Had he not extended his encouragement, support, and understanding, I never would have made it through the field of personal crisis that arose during my academic tenure. His expertise was an invaluable asset to me.

I would also like to thank Dr. Donald Kraft for also being so supportive and encouraging during the dramatic changes that occured in my life duing graduate school. His wit (and dare I say it, his madness) put a smile on my face on many occasions. He has been an inspriation to me. He is a true friend and mentor.

Dr. Sithirama Iyengar, Dr. Doris Carver, and Dr. Harriet Taylor have all inspired me to excell. They have all three put forth an effort to make opportunities available for me that otherwise would have gone unexplored. Thank you for giving me the chance to prove myself.

I would like to offer a special thanks to my committee members Dr. J. Bush Jones, Dr. Donald Kraft, Dr. John Tyler, Dr. Sithirama Iyengar, Dr. Doris Carver, Dr. Robert Dorroh, and Dr. Paul Kirk whose supervision of my research inspired me to excellence.

I would like to extend a special word of thanks to Luegina Mounfield. Without her encouragement, I may not have changed my major to Computer Science and found what a truly wonderful and exciting discipline Computer Science is.

How do I find the words to express my appreciation to the single greatest inspiration in my life? To my father, Harold D. Iles, who taught me to be the best

that I can be, to do the best that I can do, and to never sacrifice my integrity, I say thanks.

To my sister and friend, Edith Soberon, I couldn't have done it without you. Thanks for letting me call you in the middle of the night and cry on your shoulder, and thanks for always being there with.and encouraging word when I needed it.

Finally, to my precious Nelson Aymond, who kept me warm, motivated, and fed with good Cajun food, I must say that your support made all of the difference in the world. Lache pas la patate!

Table of Contents

List of Figures

Abstract

Bush Jones published a series of papers providing sequential algorithms that are key to reconstructability analysis. These algorithms include the determination of unbiased reconstructions and a greedy algorithm for a generalization of the reconstruction problem. The implementation of these sequential algorithms provide scientists and mathematicians with the means of utilizing reconstructability analysis in systems modeling. The algorithms, however, are so computationally intensive that the system is limited to a very small set of variables.

Many papers have been written applying reconstructability analysis and maximum entropy methods to various disciplines. Reconstructability analysis has the potential of dramatically impacting the scientific community, but the sequential algorithms leave the utilization of reconstructability analysis infeasible. The author has parallelized the reconstructability analysis algorithms developed by Jones, thereby, bridging the gap between theoretical application and feasible implementation.

Since the goal of parallelization of these reconstructability analysis algorithms is to make them feasible to as many researchers as possible, a specific architecture is not assumed. It is assumed that the architecture employed is a multiple data architecture. That is, the architectural design needed for the implementation of these algorithms must have memory local to each processing element (PE). The parallel algorithms developed and presented here do not address the problems of communications between processors of particular architectures. These algorithms assume a reconfigurable bus system which is a bus system whose configuration can be

dynamically altered thus allowing broadcasting and long-distance communications to be completed in constant time. It is noted that processor arrays with such reconfigurable bus systems have been designed.

Frequently, parallel algorithms do not address the situation in which the number of values on which to operate is larger than the number of processors. However, since the purpose of the parallelization of these reconstructability analysis algorithms is to make them feasible for large structure systems, the parallelization given does address the situation in which the number of values on which to operate is larger than the number of processors available. Therefore, implementation of the algorithms involves simply incorporating the communication protocols between processors for the particular architecture employed.

Chapter 1

Overview of Reconstructability Analysis

Traditionally, systems modeling focuses on the validity of subsystems. In the early 1960's Ross Ashby broke with tradition and considered the reconstructability and validity of the overall system. The mid 1970's saw more comprehensive investigations of the problem of reconstructability. Reconstructability analysis sprang to life in the early 1980's when the International Journal of General Systems devoted a special issue to the evaluation of the reconstruction hypothesis. [CAVA 81a] In this special issue, Roger Cavallo and George Klir present a comprehensive evaluation of the reconstruction hypothesis. The main results of the paper are that

> "(1) the concept of meaningful reconstruction hypothesis is developed; (2) efficient procedures are derived for determining the unbiased reconstruction for any given reconstruction hypothesis; (3) the concept of the reconstruction family is developed; (4) efficient procedures are proposed for determining the reconstruction family for any given structure system." [CAVA 81b]

In the mid 1980's Bush Jones published a series of papers providing sequential algorithms that are key to reconstructability analysis. These algorithms include the determination of unbiased reconstructions [JONE 85a] and a greedy algorithm for a generalization of the reconstruction problem [JONE 85b]. The implementation of these sequential algorithms provide scientists and mathematicians with the means of utilizing reconstructability analysis in systems modeling. The algorithms, however, are so computationally intensive that the system is limited to a very small set of variables.

Many papers have been written applying reconstructability analysis and maximum entropy methods to various disciplines and sub-disciplines. Reconstructability analysis has the potential of dramatically impacting the scientific community, but the sequential algorithms leave the utilization of reconstructability analysis infeasible. The author has parallelized the reconstructability analysis algorithms developed by Jones [JONE 85a] [JONE 85b], thereby, bridging the gap between theoretical application and feasible implementation.

1.1 Introduction

1.1.1 General Definitions

Cavallo and Klir describe two problems in systems modeling: the reconstructability problem and the identification problem. The reconstructability problem is viewed as the determination of whether a subsystem is adequate to reconstruct the behavior of the overall system within a desired level of approximation. The identification problem is viewed as the determination of properties of an unknown overall system from appropriate properties of given subsystems. The process of solving these two problems is called reconstructability analysis [CAVA 81a].

Cavallo and Klir, pioneers in reconstructability analysis, define a behavior system as a sixtuple:

$$"B = (V, \mathscr{V}, s, A, Q, f) \qquad (1)$$

where $V = \{v_i | i \in N_l\}$ ($N_l = \{1,2,...,l\}$) is a set of variables, $\mathscr{V} = \{V_j | j \in N_m, m \leq l\}$ is a family of state sets; $V \rightarrow \mathscr{V}$ is an onto assignment function by which one state set from $\mathscr{V}$ is assigned to each variable in V; $A = s(v_1) \times s(v_2) \times ... \times s(v_l)$ is the set of all potential aggregate states; Q is a set of real numbers which includes 0; $f: A \rightarrow Q$ is a function, referred to as a behavior function, which represents

information regarding the aggregate states of the behavior system" [CAVA 81b].

(Note that the author has taken the liberty of changing variables in the definition to avoid notation confusion in future chapters.) A behavior function may be considered as an overall system [CAVA 81b]. An example of a probabilistic behavior function is illustrated in Figure 1.

Before proceeding further, some notation must be defined:

"For each aggregate state (n-tuple)

$$\alpha = (\alpha_i \mid i \in N_l) \in A$$

of a behavior system defined by (1) and for each state

$$\beta = (\beta_j \mid j \in X, X \in l)$$

associated with variables in set

$$Z = \{v_j \mid j \in X, X \subset N_l\} \subset V,$$

let β be called a substate of α (or α be called a superstate of β) if and only if

$$\beta_j = \alpha_j \text{ for all } j \in X.$$

Let $\beta\langle\alpha$ (and $\alpha\rangle\beta$) denote that β is a substate of α" [CAVA 81b].

"Let $[f\downarrow Z]$ denote the projection of f which disregards all variables in V except those in set $Z \subset V$. Then, $[f\downarrow Z]$ is itself a mapping from a set of states (substates of states in A) to Q ... such that

$$[f\downarrow Z](\beta) = g(\{f(\alpha) \mid \alpha\rangle\beta\}),$$

where function g is determined by the nature of function f. For instance,

$$[f\downarrow Z](\beta) = \sum_{\alpha\rangle\beta} (f(\alpha))$$

when f is a probabilistic function" [CAVA 81b].

Figure 2 illustrates a projection.

A behavior function may also be considered as a subsystem of a larger behavior system [CAVA 81b]. Given an overall behavior system B as defined above, another system

$$^0B = (^0V, {}^0\mathcal{T}, {}^0s, {}^0A, {}^0Q, {}^0f)$$

v_0	v_1	v_2	f( )
0	0	0	0.0
0	0	1	0.2
0	1	0	0.1
0	1	1	0.3
1	0	0	0.0
1	0	1	0.1
1	1	0	0.2
1	1	1	0.1

Figure 1: A Probabilistic Behavior Function

If $Z = \{v_2, v_3\}$ and f( ) is the behavior function as defined in Figure 1, then $[f \downarrow Z]$, the projection of f is a mapping from {00, 01, 10, 11} to [0, 1] where

$$[f \downarrow Z](00) = f(000) + f(100) = 0.0$$
$$[f \downarrow Z](01) = f(001) + f(101) = 0.3$$
$$[f \downarrow Z](10) = f(010) + f(110) = 0.3$$
$$[f \downarrow Z](11) = f(011) + f(111) = 0.4$$

Figure 2: A Projection

is a subsystem B if and only if it satisfies the following conditions:

"* ${}^0V \subset V$;
* ${}^0\mathscr{V} \subseteq \mathscr{V}$ such that ${}^0s$ is onto;
* ${}^0s : {}^0V \rightarrow {}^0\mathscr{V}$ such that ${}^0s(v_i) = s(v_i)$ for each $v_i \in {}^0V$;
* ${}^0A = \prod {}^0s(v_i)$ such that $v_i \in {}^0V$;
* ${}^0Q = Q$;
* ${}^0f = [f\downarrow {}^0V]$" [CAVA 81b].

The system illustrated in Figure 3 is a subsystem of the system in Figure 1.

> "Let a set of behavior systems, say the set
> $$S = \{{}^kB = ({}^kV, {}^k\mathscr{V}, {}^ks, {}^kA, {}^kQ, {}^kf) \mid k \in N_q\},$$
> be referred to as a structure system. Let behavior systems ${}^kB$ be called elements of the structure system S" [CAVA 81b].

Jones clarifies this definition by stating that given an overall system B with a behavior function f and a subsystem ${}^kB$, the behavior function ${}^kf$ must satisfy the condition:

$${}^kf(\beta) = \sum_{\alpha \rangle \beta} f(\alpha),$$

where of course $\beta \in {}^kA$, $\alpha \in A$, and $\alpha \rangle \beta$ (i.e. that β is a substate of α) [JONE 82]. Note the similarity to the behavior of marginal probabilistic functions in Statistical Analysis.

1.1.2 The Unbiased Reconstruction

Now that the basic foundation has been laid in reconstructability analysis terminology, we can turn our attention to the specifics of The unbiased reconstruction. If we have a structure system $S = \{{}^kB\}$, each solution to a set of equations satisfying certain constraints defines a behavior function which "uniquely represents a member of $\mathscr{F}_s$, the reconstruction family of S" [JONE 82]. The unique member of the $\mathscr{F}_s$

which is the maximum entropy solution is called the unbiased reconstruction [JONE 85a].

The goal set before us is to determine the unbiased reconstruction with the information that is available but from nonredundant sets of information. Given the subsystems in Figure 4, an overall system can be reconstructed, but these subsystems contain redundant information. The same overall system can be reconstructed from the information in the subsystems illustrated in Figure 5. The process of eliminating redundant information involves partitioning the substate instances into disjoint independent sets and then applying the unbiased reconstruction to one of these sets. Since the unbiased reconstruction is unique, it will be the same regardless which set of states is used in the reconstruction [JONE 85a]. Jones suggests generalizing the reconstruction problem by partitioning the substates into disjoint equivalence classes by using the concept of null extensions:

> "an overall state α is the null extension of β, written β', if $\alpha \rangle \beta$ and every variable of α which does not occur in β has a value of zero" [JONE 85b].

In 1964, Ross Ashby suggested a procedure for determining the unbiased reconstruction for a given reconstruction hypothesis. The Ashby procedure can be expressed by the formula

$$F_s = \bigcap_{k \in N_q} [{}^kF \uparrow V -^k V]$$

where F denotes the behavior relation of an overall system B and ${}^kF$ denotes the behavior relation of the selection behavior function ${}^kf = [f \downarrow^k V](k \in N_q)$ involved in the reconstruction hypothesis S. $[{}^kF \uparrow V -^k V]$ is the extension of ${}^kF$ with respect to

v_0	v_1	$^2f(\,)$
0	0	0.2
0	1	0.4
1	0	0.1
1	1	0.3

Figure 3: A Subsystem of the System in Figure 1

v_1	v_2	$^4f(\,)$	v_2	v_3	$^5f(\,)$	v_1	v_3	$^6f(\,)$
0	0	0.25	0	0	0.17	0	0	0.11
0	1	0.18	0	1	0.16	0	1	0.14
1	0	0.20	0	2	0.12	0	2	0.18
1	1	0.37	1	0	0.14	1	0	0.20
			1	1	0.18	1	1	0.20
			1	2	0.23	1	2	0.17

Figure 4: A Set of Subsystems

v_1	v_2	$^4f(\,)$	v_2	v_3	$^5f(\,)$	v_1	v_3	$^6f(\,)$
1	1	0.37	1	0	0.14	0	1	0.14
			1	1	0.18	0	2	0.18
			1	2	0.23	1	0	0.20
						1	1	0.20
						1	2	0.17

Figure 5: Subsystems with Nonredundant Information

the variables V-^kV [CAVA 81b]. One way to implement the Ashby procedure is to use the complement of the equation:

$$\bar{F}_s = \bigcup_{k \in N_q} \left[{}^k\bar{F} \uparrow V - {}^kV \right]$$

where ${}^k\bar{F} = {}^kA - {}^kF$. This complement is then used in the following algorithm [CAVA 81b]

> "Given a reconstruction hypothesis S with behavior relations ${}^kF(k \in N_q)$, to determine the unbiased (largest) behavior relation F_s implied by S:
> [1] Let $X = A$ and let $k = 1$;
> [2] for each $\alpha \in X$ and each $\beta \in {}^k\bar{F}$, if $\beta \langle \alpha$, then make $X - [\alpha] \rightarrow X$;
> [3] if $k < N_q$, make $k + 1 \rightarrow k$ and go to [2];
> [4] stop; $F_s = X$" [CAVA 81b].

Cavallo and Klir give an algorithm for the determination of the unbiased reconstruction based on "the use of the relational join and forms the overall maximum entropy relation F_s by sequentially joining each of the relations associated with the subsystem" [CAVA 81b].

Jones revised the algorithm taking into account the partitioning the substates into equivalence classes. Jones' revision eliminates the requirement of a complete set of substates for the structure system. Jones' unbiased reconstruction algorithm works for an arbitrary collection of substates [JONE 85a]. Even though his general algorithm is short and concise, implementation is not trivial. This algorithm is presented in detail in section 1.2.1.

As you may have noticed, the discussion up to this point has considered only behavior functions that are probabilistic and that have discrete data values. Jones has

extended the reconstruction process to the application on both general functions - functions that are not necessarily probabilistic or possibilistic behavior functions - and arbitrary data.

The general function, called a g-system, is first transformed to a dimensionless form and then "a mathematical structure is then induced via a type of isomorphism onto this system which renders it amenable to analysis by established techniques" [JONE 85c]. The system induced by this isomorphism is called a *k*-system or Klir system. A *g*-system is formally defined as a sixtuple:

$$"(\tau, \{v_i\}, \{\alpha\}, \{\beta\}, f(\bullet), \{^m f(\bullet)\})$$

* τ is a parameter
* $\{v_i\}$ is a set of variables;
* $\{\alpha\}$ is a set of states;
* $\{\beta\}$ is a set of substates;
* $f(\bullet)$ is a function on $\{\alpha\}$;
* $\{^m f(\bullet)\}$ are functions on $\{\beta\}$" [JONE 85c].

The resulting *k*-system is, of course, also a sixtuple:

$$"(\tau, \{v_i\}, \{\alpha\}, \{\beta\}, k(\bullet), \{^m k(\bullet)\})$$

* τ is a parameter;
* $\{v_i\}$ is a set of variables;
* $\{\alpha\}$ is a set of states;
* $\{\beta\}$ is a set of substates;
* $f(\bullet)$ is a function on $\{\alpha\}$;
* $\{^m f(\bullet)\}$ are functions on $\{\beta\}$" [JONE 85c].

Since a *k*-system is isomorphic to a *g*-system, there is no information present in the *k*-system which is not present in the *g*-system and all of the information in the *g*-system is present in the *k*-system. One can, therefore, apply reconstructability analysis to a *k*-system to deduce or discover information about the isomorphic *g*-system [JONE 85c].

Jones has also extended the reconstruction process to systems of arbitrary data. Three types of problems arise when dealing with arbitrary data: data scattering, state contradictions, and missing data. These problems are addressed by clustering data, refining the variables or introducing new variables, and considering only known states, respectively [JONE 85d]. Therefore, we can treat general systems with arbitrary data as the simpler probabilistic behavior functions with discrete data values without loss of generality.

1.1.3 Maximum Entropy

Reconstructability Analysis employs the principle of maximum entropy to ensure that no extraneous information is factored into the reconstruction process. First, we will look at the properties of entropy, then we will examine the principle of maximum entropy.

Entropy itself is "considered a good measure of the amount of uncertainty contained in a probability distribution" [GUIA 85]. Consider the probability distribution $\bar{p} = (p_1, p_2, \cdots, p_m)$ which of course satisfies the conditions $p_k \geq 0$ and $\sum_{k=1}^{m} p_k = 1$. The entropy of the probability distribution is the number "$H_m(\bar{p}) = H_m(p_1, p_2, \cdots, p_m) = -\sum_{k=1}^{m} p_k \ln p_k$ " [GUIA 85]. We define $0 \ln 0 = 0$ to ensure the continuity of the function at the origin.

Entropy displays some interesting properties. First, entropy is continuous, invariant under any permutation of the indices, and $H_m(\bar{p}) \geq 0$. Second, if a probabilistic experiment has only one possible outcome, then it contains no

uncertainty. In other words, $H_m(1) = 0$. Third, if we have a probabilistic experiment with a set number of possible outcomes, and we add another outcome with the probability of 0, the amount of uncertainty in the experiment does not change: $H_m(p_1, p_2 \cdots p_m) = H_{m+1}(p_1, p_2, \cdots p_m, 0)$. Fourth, the maximum uncertainty is contained in the uniform distribution. $H_m(p_1, p_2, \cdots, p_m) \le H_m\left(\frac{1}{m}, \frac{1}{m}, \cdots \frac{1}{m}\right)$ and $H(p_1, p_2, \cdots p_m) = H_m\left(\frac{1}{m}, \frac{1}{m}, \cdots \frac{1}{m}\right)$ if $p_k = \frac{1}{m}\ \forall 1 \le k \le m$. Fifth, the amount of uncertainty in a pair of random variables is the sum of the amount of uncertainty in one variable and the uncertainty in the other variable conditioned by the first variable. In other words $H(X, Y) = H(X) + H(Y|X)$. If $\bar{\pi} = (\pi_{1,1}, \ldots, \pi_{m,n})$ is a joint probability distribution with marginal probability distributions $\bar{p} = (p_1, \ldots, p_m)$ and $\bar{q} = (q_1, \ldots q_n)$, then $H(X, Y) = H_{mn}(\pi_{1,1} \ldots, \pi_{m,n})$, $H(X) = H_m(p_1, \ldots p_m)$, and $H(Y|X) = \sum_{k=1}^{m} p_k H_n\left(\frac{\pi_{k,1}}{p_k}, \ldots, \frac{p_{k,n}}{p_k}\right)$. Therefore, $H_{m,n}(\pi_{1,1}, \ldots, \pi_{m,n}) = H_m(p_1, \ldots, p_m) + \sum_{k=1}^{m} p_k H_n\left(\frac{\pi_{k,1}}{p_k}, \ldots, \frac{\pi_{k,n}}{p_k}\right)$. Note that $H_n\left(\frac{\pi_{k,1}}{p_k}, \ldots \frac{\pi_{k,n}}{p_k}\right)$ is only computed for those values of k for which $p_k \neq 0$. Sixth, "some data on X can only decrease the uncertainty on Y, namely $H(Y|X) \le H(X) + H(Y)$ with equality if and only if X and Y are independent" [GUIA 85]. In other words, $\sum_{k=1}^{m} p_k H_n\left(\frac{\pi_{1,n}}{p_k}, \ldots, \frac{\pi_{k,n}}{p_k}\right) \le H_n(q_1 \ldots q_n)$ with equality if and only if $\pi_{k,l} = p_k q_l\ \forall 1 \le k \le m$ and $\forall 1 \le l \le n$. Note that when there is equality, $H_{m,n}\left(\bar{\pi}\right) = H_m\left(\bar{p}\right) + H_n\left(\bar{q}\right)$. [GUIA 85]

According to Laplaces' Principle of Insufficient Reason, if we have no reason to discriminate between events, the best strategy is to consider them equally likely.

Laplaces' Principle is simply a point of view based on common sense, but it describes the fourth property of entropy listed above. According to this property, uncertainty is maximized when the outcomes are equally likely. In other words, "the uniform distribution contains the largest amount of uncertainty"; "the uniform distribution maximizes entropy" [GUIA 85]. The Principle of Maximum Entropy asserts that "entropy is maximized by the uniform distribution when no constraint is imposed on the probability distribution" [GUIA 85]. When constraints are imposed on the distribution, we want to find the "best" probability distribution that is compatible with the set of known mean values of one or more random variables. E. T. Jaynes suggested the criterion to which to base this "best" choice by introducing the Principle of Maximum Entropy in 1957. We simply choose the distribution that maximizes Shannon's entropy $\left(H_m\left(\bar{p}\right) = H_m(p_1, \ldots p_m) = -\sum_{k=1}^{m} p_k \ln p_k\right)$. This probability distribution will "ignore no possibility, being the most uniform one, subject to the given constraint" [GUIA 85]. The Principle of Maximum Entropy is used in many applications because what is generally known is expressed by mean values of some random variables and what is needed is a probability distribution that ignores no possibility subject to the constraints on the system. [GUIA 85]

There are several remarkable results relating to the Principle of Maximum Entropy that are mentioned here:

> "a) If f is a random variable whose range is countable, namely, if
> $$\{ku | u > 0, k = 0, 1, 2, \ldots\}$$
> ..., and if the mean value $E(f)$ is given , then the probability distribution

$$p_k > 0, (k = 0, 1, \ldots), \sum_{k=0}^{\infty} p_k = 1$$

maximizing the countable entropy:

$$H = -\sum_{k=0}^{\infty} p_k \ln p$$

is

$$p_k = \frac{u(E(f))^k}{(u+E(f))^{k+1}}, k = 0, 1, 2, \ldots$$

We see that the unit u and the mean value $E(f)$ completely determine the solution of the Principle of Maximum Entropy" [GUIA 85].

"b) In the continuous case, suppose that we know the mean value μ of a positive continuous random variable whose probability density function is square-integrable. In such a case, the continuous entropy

$$H(\delta) = -\int_{-\infty}^{+\infty} \delta(x) \ln \delta(x) dx$$

is maximized by

$$\delta(x) = \left\{ \begin{array}{c} \frac{1}{\mu} e^{-\frac{1}{\mu}x}, \text{ if } x > 0 \\ 0, \text{ elsewhere} \end{array} \right\}$$

which is just the well known exponential probability density function" [GUIA 85].

"c) Of course, it is possible to have many constraints. Suppose that, in the continuous case, we know both the mean μ and the variance σ^2 of a continuous random variable whose probability density function is square-integrable. ... In such a case, the continuous entropy is maximized just by

$$\delta(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, (-\infty < x < +\infty)$$

which is the probability density function of the normal distribution $N(\mu, \sigma^2)$" [GUIA 85].

1.2 Sequential Algorithms

1.2.1 Determination of Unbiased Reconstructions

Jones' algorithm for the determination of unbiased reconstructions is broken into three steps: generating independent substates, partitioning the states into disjoint sets, and the reconstruction process. These sequential algorithms are given below. Note that through both the explanation of the sequential and parallel algorithms, the

sample structure system will the one illustrated in figure 6. This example structure system may be found in [CAVA 81b], [JONE 85a], and [JONE 85b].

1.2.1.1 Independent Substates

Frequently, the state information given for a structure system contains redundant information. As an example, the same unbiased reconstruction can be obtained from the structure system in figure 7 that is obtained in figure 6. It is desirable to determine an unbiased reconstruction from as few states as possible (that is from nonredundant information sets). There are several advantages to being able to work with limited or independent information. One is space considerations. Another advantage is the amount of time saved in computing the reconstruction family and the unbiased reconstruction. We also have the advantage of only needing to collect information on the independent states. Some state information may be costly or difficult to obtain.

The set of independent substates is obtained by partitioning the substates into equivalence classes based on the concept of null extensions. The independent states are then obtained by choosing exactly one substate from each equivalence class. Since any one substate from each equivalence class is sufficient for the construction of a set of independent substates, this set is not unique.

1.2.1.1.1 Equivalence Classes

Before we consider Jones' greedy algorithm for the generalization of the reconstruction problem, we need to consider the partitioning of the substates into disjoint equivalence classes. . By definition, a binary relation $\sim$ on a set X is an

v_2	v_3	$^5f(\,)$
0	0	0.17
0	1	0.16
0	2	0.12
1	0	0.14
1	1	0.18
1	2	0.23

v_1	v_3	$^6f(\,)$
0	0	0.11
0	1	0.14
0	2	0.18
1	0	0.20
1	1	0.20
1	2	0.17

v_1	v_2	$^4f(\,)$
0	0	0.25
0	1	0.18
1	0	0.2
1	1	0.37

Figure 6: Example Structure System

v_2	v_3	$^5f(\,)$
1	0	0.14
1	1	0.18
1	2	0.23

v_1	v_3	$^6f(\,)$
0	1	0.14
0	2	0.18
1	0	0.20
1	1	0.20
1	2	0.17

v_1	v_2	$^4f(\,)$
1	1	0.37

Figure 7: Example Structure System with Only Nonredundant Information

equivalence relation if the following three conditions hold:

(1) $\sim$ is reflexive: $x \sim x \ \forall x \in X$

(2) $\sim$ is symmetric: $x \sim y \ \forall y \sim x$

(3) $\sim$ is transitive: $x \sim y$ and $y \sim z \Rightarrow x \sim z$

The equivalence class of the element $x \in X$ is the set $[x] = \{y \in X : x \sim y\}$. A partition of the X is a collection of nonempty subsets $\{A_i\}$ of X such that $A_i \cap A_j = \varnothing$ if $i \neq j$ and $X = \cup_i A_i$.

Claim: If $\sim$ is an equivalence relation on the set X, the collection of all equivalence classes is a partition of X.

Proof: By definition of equivalence classes, $x \in [x] \ \forall x \in X$. Therefore, the equivalence classes are subsets of X such that $X = \bigcup_{x \in X} [x]$. Now we must show that the equivalence classes are disjoint. If $[x] \cap [y] \neq \varnothing$, then there is an element $z \in [x] \cap [y]$. Therefore, $x \sim z$ and $y \sim z$. Since $\sim$ is symmetric, $z \sim y$. Therefore, $x \sim z$ and $z \sim y$. Since $\sim$ is transitive, $x \sim y$. Therefore, $y \in [x]$ and $[y] \subset [x]$. By symmetry of the set operation intersection, if $z \in [x] \cap [y]$ then $z \in [y] \cap [x]$. Therefore $y \sim z$ and $x \sim z$. Since $\sim$ is symmetric, $z \sim x$. Therefore , $y \sim z$ and $z \sim x$. Since $\sim$ is transitive , $y \sim x$.Therefore, $x \in [y]$ and $[x] \subseteq [y]$. Since $[y] \subseteq [x]$ and $[x] \subseteq [y]$, then $[x] = [y]$. Therefore, the equivalence classes are disjoint, and therefore form a partition of the set X.

Claim: If $\{A_i\}$ is a partition of the set X, then there is an equivalence relation on X whose equivalence classes are the subsets A_i.

Proof: We define the relation $\sim$ on the set X as follows: $x \sim y$ if and only if x and y are

in the same subset A_i. We must show that the three properties of an equivalence relation hold. Clearly the reflexive and symmetric properties hold. Suppose that $x \sim y$ and $y \sim z$. Then x and y are in the same subset A_i and y and z are in the same subset. Clearly, the two subsets must be the same since the all of the subsets A_i of a partition are disjoint.

Therefore, $\sim$ is necessarily an equivalence relation. Suppose $x \in A_i$, then $x \sim y$ if and only if $y \in A_i$. Therefore, $[x] = A_i$. [ADKI 92]

1.2.1.1.2 Null Extensions

Null extensions can be used for the partitioning process. Let β be a substate of a system, and let α be a state in the overall system. We say α is the null extension of β, written β', if "every variable of α which does not occur in has a value of zero" [JONE 85b]. Figure 6 illustrates the null extensions of given substates.

"Two substates, β_i and β_j, are said to be equivalent if $\beta_i' = \beta_j'$, and thus both are members of the same equivalence class" [JONE 85b]. The overall system for the subsystems given in figure 8 is given in figure 9. Using the notation such that , substate ($v_i = x$) is denoted by $^{i}(x)$, substate ($v_j = y$, $v_k = z$) is denoted by $^{jk}(y,z)$, etc., figure 10 illustrates the equivalence classes for the system given in figure 9. Since the condition has been set that a distribution must sum to one, the class E_{10} is not necessary in our analysis.

			Null Extension		
v_2	v_3	$^5f(\,)$	v_1	v_2	v_3
0	0	0.17	0	0	0
0	1	0.16	0	0	1
0	2	0.12	0	0	2
1	0	0.14	0	1	0
1	1	0.18	0	1	1
1	2	0.23	0	1	2

			Null Extension		
v_1	v_3	$^6f(\,)$	v_1	v_2	v_3
0	0	0.11	0	0	0
0	1	0.14	0	0	1
0	2	0.18	0	0	2
1	0	0.20	1	0	0
1	1	0.20	1	0	1
1	2	0.17	1	0	2

			Null Extension		
v_1	v_2	$^4f(\,)$	v_1	v_2	v_3
0	0	0.25	0	0	0
0	1	0.18	0	1	0
1	0	0.2	1	0	0
1	1	0.37	1	1	0

Figure 8: Null Extensions of Substates

v_1	v_2	v_3	f( )
0	0	0	0.079
0	0	1	0.088
0	0	2	0.083
0	1	0	0.031
0	1	1	0.052
0	1	2	0.097
1	0	0	0.091
1	0	1	0.072
1	0	2	0.037
1	1	0	0.109
1	1	1	0.128
1	1	2	0.133

Figure 9: Overall System

$E_1 = \{ {}^1(1), {}^{12}(10), {}^{13}(10) \}$
$E_2 = \{ {}^2(1), {}^{12}(01), {}^{23}(10) \}$
$E_3 = \{ {}^3(1), {}^{13}(01), {}^{23}(01) \}$
$E_4 = \{ {}^3(2), {}^{13}(02), {}^{23}(02) \}$
$E_5 = \{ {}^{12}(11) \}$
$E_6 = \{ {}^{13}(11) \}$
$E_7 = \{ {}^{23}(11) \}$
$E_8 = \{ {}^{13}(12) \}$
$E_9 = \{ {}^{23}(12) \}$
$E_{10} = \{ {}^1(0), {}^2(0), {}^3(0), {}^{12}(00), {}^{13}(0,0), {}^{23}(0,0) \}$

Figure 10: Equivalence Classes

1.2.1.1.3 Generate Independent Substates

A set of independent substates is generated into a set D as follows. The set D is initially empty. A substate is arbitrarily chosen to be added to the set D. While more independent states are desired, another substate is arbitrarily chosen. If its null extension is not the same as any of the substates in the set D then that substate is added

to the set D. When the algorithm terminates iterating, the set D contains only independent substates.

The algorithm as stated by Jones is as follows:

> "To generate independent substates of a system into a set D, do the following.
> 1) Let $\beta_1 \in D$ where β_1 is any substate, $i \leftarrow 2$.
> 2) Let $\beta_i \in D$ if β_i is such that $\beta_i' \neq \beta_j'$ for $j = i - 1, i - 2, ..., 1$.
> 3) $i \leftarrow i + 1$, If more independent states are desired go to [2]; else stop" [JONE 85a].

Now that the set of state information has been reduced to the set of independent states, the remaining steps of the unbiased reconstruction may be employed with the use of only nonredundant information.

1.2.1.2 Partition the States into Disjoint Sets

The second step in the determination of unbiased reconstruction is the partitioning of the states into disjoint sets. Although Jones found that in practice the algorithm works without the partitioning step, the reconstruction process step has been found to converge faster if the partition is employed [JONE 85a].

The set of substates are partitioned into disjoint sets such that no two elements in a partition set have a mutual superstate. The partition is performed by initially

taking an arbitrary element from the set of substates and inserting it into a partition set C_1. Each remaining substate is added to C_1 one at a time only if it does not have a superstate that is the superstate of an element of C_1. Once all of the remaining substates have been chosen that can be chosen for this partition set, the next partition set is constructed in the same manner. This partitioning process continues until there are no more substates to be partitioned:

> "Let $\{ \beta_i \}$, i = 1,2,... ,n, be an arbitrary collection of substates of an overall system. We partition $\{ \beta_i \}$ into disjoint sets C_1 , C_2 ,...,C_m.
>
> A) First, to form C_1 do the following.
> 1) Let $\beta_1 \in C_1; i \leftarrow 2$
> 2) If there exists no $\alpha \rangle \beta_i$ such that for some $\beta_j \in C_1$ then let $\beta_i \in C_1$.
> 3) $i \leftarrow i + 1$, if $i \leq n$ go to [2]; else C_i formed.
>
> B) C_2 is formed in a similar manner from the β_i not placed in C_1.
>
> C) Form C_3 ,..., C_m similarly until no β_i remain" [JONE 85a].

Note that since the selection of the substates to add to a partition is arbitrary, the partition is not unique.

Once the partition is complete, we associate the equation

$$\sum_{\alpha \rangle \beta_i} f(\alpha) = {}^k f(\beta_i)$$

to each element of the partition. For each C_i we add one equation to the set of the form

$$"\Sigma_1 f(\alpha) = 1 - \Sigma_2 f(\beta_i)$$

where Σ_2 is over the β_i of C_i, and ${}^k f(\bullet)$ depends on the particular β_i. Σ_1 is over α such that $\alpha \rangle \beta_j$ for some $\beta_j \in C_i$ is not true" [JONE 85a]. Note that we can write these equations as

$$\sum_j f_{ij} = a_i \; \forall a_i$$

where a_i is the probability of a state in a partition and the f_{ij} are the appropriate left hand side of the equation. Figure 11 displays a set of equations for the example structure system using the independent states given in figure 7. These equations can be considered as those of a reconstruction hypothesis.

1.2.1.3 Reconstruction Process

Now that we have a reconstruction hypothesis in the form

$$\sum_j f_{ij} = a \text{for all } a_i,$$

we can obtain an unbiased reconstruction. Note that we denote an approximation to f_{ij} by $\hat{f}_{ij}$ and we let $\hat{a}_i = \Sigma_j \; \hat{f}_{ij}$. The unbiased reconstruction can be completed by following steps:

"1) Initialize $\hat{f}_{ij}$ to a flat distribution.

2) For all i:

$$\text{new}\hat{f}_{ij} = \hat{f}_{ij} \frac{a_i}{\hat{a}_{ij}} \text{ for every } j.$$

3) Convergence test:

$$\left| \text{new}\hat{f}_{ij} - \text{old}\hat{f}_{ij} \right| < \varepsilon \quad \text{for every } i,j.$$

If not converged, go to (2); else stop " [JONE 85b].

This algorithm applied to the example given in figure 6 yields the overall system given in figure 12. Note that this result is the same overall system obtained by Jones when applied to Klir and Cavallo's algorithm on the information in its redundant form [JONE 85a].

C_1

$$f(110)+f(111)+f(112)={}^{4}f(11)$$
$$f(001)+f(011)={}^{6}f(01)$$
$$f(002)+f(012)={}^{6}f(02)$$
$$f(000)+f(010)+f(100)+f(101)+f(102)=1-{}^{6}f(02)-{}^{4}f(11)-{}^{6}f(01)$$

C_2

$$f(100)+f(110)={}^{6}f(10)$$
$$f(011)+f(111)={}^{5}f(11)$$
$$f(012)+f(112)={}^{5}f(12)$$
$$f(000)+f(010)+f(001)+f(101)+f(002)+f(102)=1-{}^{6}f(10)-{}^{5}f(11)-{}^{5}f(12)$$

C_3

$$f(010)+f(110)={}^{5}f(10)$$
$$f(101)+f(111)={}^{6}f(11)$$
$$f(102)+f(112)={}^{6}f(12)$$
$$f(000)+f(100)+f(002)+f(001)+f(011)+f(012)=1-{}^{5}f(10)-{}^{6}f(11)-{}^{6}f(12)$$

Figure 11: Set of Equations for Partition of Independent States

v_1	v_2	v_3	f( )
0	0	0	0.079
0	0	1	0.088
0	0	2	0.083
0	1	0	0.031
0	1	1	0.052
0	1	2	0.097
1	0	0	0.091
1	0	1	0.072
1	0	2	0.037
1	1	0	0.109
1	1	1	0.128
1	1	2	0.133

Figure 12: Example Overall System

1.2.2 A Greedy Algorithm for a Generalization of the Reconstruction Problem

1.2.2.1 The Greedy Algorithm

The generalization of the Reconstruction problem removes the restriction that the reconstruction must be to a structure system. Substates in general are allowed in the reconstruction, and the overall system does not necessarily need to be known in its entirety. The unbiased reconstruction is computed for the limited information, and then we can proceed as if the overall system is known along with its probabilistic behavior functions f and ${}^{k}f$.

Let E be the set of nonredundant substates, let D be the set of known substates to be used in the reconstruction, let $U(D)$ represent the unbiased reconstruction on the set D, and let $\gamma(\bullet)$ be a choice function for substates where $\gamma(\beta)$ measures the desirability of β for inclusion in the reconstruction set D. Jones gives the general form of the algorithm as follows:

> "ALGORITHM Given knowledge of an overall behavior function $f_{i,j}$ and hence all ${}^{k}f$ for substates, and the set E, determine the reconstruction set $D \subset E$ as follows.
>
> i) Initialization: initialize $\hat{f}_{ij}$ to a flat distribution: Let D be initially empty.
>
> ii) Selection of one β to add to D:
>
> $$\beta = \arg\max_{\beta \in E} \gamma(\beta)\left(\gamma(\bullet)\text{makes use of the current } \hat{f}_{ij}\right)$$
>
> Let $\beta \in D$; $E = E - D$ (remove β from E)
>
> iii) Compute unbiased reconstruction for new D: $U(D) \rightarrow \hat{f}_{ij}$
> (Note: in computing $U(D)$ the previous unbiased reconstruction may be taken as the initialization; this greatly hastens convergence.)

iv) Stopping rule:
Size limit for number of members in D exceeded?
or
$\left| f_{ij} - \hat{f}_{ij} \right| < ?$ (where the norm is information distance)
Yes, stop.
No, go to (ii)" [JONE 85b].

Figure 13 shows the iterations of the unbiased reconstruction on the example structure system. Note that the algorithm converges to the example system given in figure 9.

1.2.2.2 The Maximizing Function

The choice of the γ function is critical. We want to use a choice function that will bring about the greatest improvement in $\hat{f}_{ij}$. According to [JONE 85b], $\hat{f}_{ij}$ improves the most when the β is added that maximizes the function

$$\gamma(\beta) = {}^{k}f(\beta)\log_2 \frac{{}^{k}f(\beta)}{{}^{k}\hat{f}(\beta)} + (1-{}^{k}f(\beta))\log_2 \frac{\left(1-{}^{k}f(\beta)\right)}{\left(1-{}^{k}\hat{f}_{ij}(\beta)\right)} \quad .$$

Jones derives this maximizing function as follows:

"Let

$$\bar{f}_{ij} = \hat{f}_{ij} \frac{{}^{k}f(\beta)}{{}^{k}\hat{f}(\beta}$$

or

$$\bar{f}_{ij} = \hat{f} \frac{\left(1-{}^{k}f(\beta)\right)}{\left(1-{}^{k}\hat{f}(\beta)\right)}$$

as appropriate for the particular f_{ij}. Consider the well known measure of closeness of $\bar{f}_{ij}$ to f_{ij}

$$\bar{I} = \Sigma f_{ij} \log \left(\frac{f_{ij}}{\hat{f}_{ij}} \right)$$

which is positive and decreases as $\bar{f}_{ij}$ approach f_{ij}. Expanding the logarithm and making a substitution for $\bar{f}_{ij}$ gives

Iteration	1	2	3	4	5	6	7	8	9
State added	$^{12}(11)$	$^{23}(10)$	$^{13}(12)$	$^{23}(11)$	$^{13}(11)$	$^{23}(12)$	$^{12}(10)$	$^{13}(01)$	$^{13}(02)$
$\hat{f}(000)$	0.07	0.073	0.076	0.085	0.088	0.086	0.083	0.081	0.079
$\hat{f}(001)$	0.07	0.073	0.075	0.085	0.088	0.086	0.083	0.088	0.088
$\hat{f}(002)$	0.07	0.073	0.076	0.085	0.088	0.086	0.083	0.081	0.083
$\hat{f}(010)$	0.07	0.048	0.045	0.034	0.032	0.033	0.032	0.032	0.031
$\hat{f}(011)$	0.07	0.073	0.076	0.044	0.049	0.05	0.05	0.052	0.052
$\hat{f}(012)$	0.07	0.073	0.076	0.085	0.088	0.097	0.097	0.097	0.097
$\hat{f}(100)$	0.07	0.073	0.076	0.085	0.088	0.086	0.092	0.092	0.091
$\hat{f}(101)$	0.07	0.073	0.076	0.085	0.069	0.07	0.07	0.072	0.072
$\hat{f}(102)$	0.07	0.073	0.055	0.042	0.039	0.037	0.037	0.037	0.037
$\hat{f}(110)$	0.123	0.092	0.095	0.106	0.108	0.107	0.108	0.108	0.109
$\hat{f}(111)$	0.123	0.139	0.16	0.136	0.131	0.13	0.13	0.128	0.128
$\hat{f}(112)$	0.123	0.139	0.115	0.128	0.131	0.133	0.133	0.133	0.133
Closeness	0.055	0.04	0.031	0.006	0.002	0.001	0.000	0.000	0.000

Figure 13: Iterations of the Greedy Algorithm

$$\bar{I} = \Sigma f_{ij} \log f_{ij} - \Sigma_1 f_{ij} \log \hat{f}_{ij} \frac{{}^k f(\beta)}{{}^k \hat{f}(\beta)} - \Sigma_2 f_{ij} \log \hat{f}_{ij} \frac{\left(1 - {}^k f(\beta)\right)}{\left(1 - {}^k \hat{f}(\beta)\right)}$$

where the sums are over appropriate terms. Now expanding the second and third logarithms

$$\bar{I} = \Sigma f_{ij} \log f_{ij} - \Sigma f_{ij} \log \hat{f}_{ij} - \Sigma_1 f_{ij} \log \frac{{}^k f(\beta)}{{}^k \hat{f}(\beta)} - \Sigma_2 f_{ij} \log \frac{\left(1 - {}^k f(\beta)\right)}{\left(1 - {}^k \hat{f}(\beta)\right)} .$$

The first two terms are simple $\hat{I}$, and taking the third and fourth sums

$$\bar{I} = \hat{I} - {}^k f(\beta) \log_2 \frac{{}^k f(\beta)}{{}^k \hat{f}(\beta)} - (1 - {}^k f(\beta)) \log_2 \frac{\left(1 - {}^k f(\beta)\right)}{\left(1 - {}^k \hat{f}(\beta)\right)} .$$

So $\hat{f}_{ij}$ improves the most when that $\beta \in E$ is added that maximizes

$$\gamma(\beta) = {}^k f(\beta) \log_2 \frac{{}^k f(\beta)}{{}^k \hat{f}(\beta)} + (1 - {}^k f(\beta)) \log_2 \frac{\left(1 - {}^k f(\beta)\right)}{\left(1 - {}^k \hat{f}(\beta)\right)} .$$

Of course, ${}^k f(\beta)$ is computed from the known overall behavior function, and ${}^k \hat{f}(\beta)$ is computed from $\hat{f}_{ij}$ " [JONE 85b].

Chapter 2

Parallel Algorithms

There are a handful of methods that can be employed to approach sequential algorithms for parallelization, but "there is no general way to parallelize sequential algorithms" [LEIG 92]. In fact, there are some sequential algorithms that cannot be parallelized at all. Therefore, we must "invent new algorithms and methods for solving problems in parallel" [LEIG 92].

Since the goal of parallelization of these reconstructability analysis algorithms is to make them feasible to as many researchers as possible, a specific architecture is not assumed. It is assumed that the architecture employed is a multiple data architecture. That is, the architectural design needed for the implementation of these algorithms must have memory local to each processing element (PE). Therefore, the parallel algorithms developed and presented here do not address the problems of communications between processors of particular architectures. In essence, these algorithms assume a reconfigurable bus system which is a bus system whose configuration can be dynamically altered thus allowing broadcasting and long-distance communications to be completed in constant time. It is noted that processor arrays with such reconfigurable bus systems have been designed.[MILL 88a][MILL 88b] [OLAR 91][STOU 86].

Frequently, parallel algorithms do not address the situation in which the number of values on which to operate is larger than the number of processors. This situation can be easily remedied by a simple conversion. Let N_1 be a network with p_1

processors, and let N_2 be a network with p_2 processors where $p_1 > p_2$. An algorithm designed for the N_1 network can be run on the N_2 network provided that N_2 is coarser-grained than N_1: Each processor of N_2 simulates $\lceil \frac{p_1}{p_2} \rceil$ processors of N_1 [LEIG 92]. However, since the purpose of the parallelization of these Reconstructability Analysis algorithms is to make them feasible for large structure systems, the parallelization given does address the situation in which the number of values on which to operate is larger than the number of processors available.

The following notations are used throughout the presentation of the parallel algorithms: (i) PE(*i*) represents the i^{th} processor; (ii) PE(*i*).X represents the variable X in the i^{th} processor's memory; (iii) A segment of the algorithm that is to be done in parallel is preceded by a *set* (in set notation) of integers ranging from 0 to p - 1 (where p is the number of processors) which designates which processors are to execute that segment of the algorithm. The parallel segment is delimited by the statements "par begin" and "par end". Note that in MIMD architectures, all processors in the set must complete this segment of the algorithm before any processor is to continue to the first statement after the "par end" statement; (iv) A critical section is denoted by the delimiters "CS begin" and "CS end". Only one processor may be executing statements in a critical section at one time. All processors of an MIMD architecture designated in the set must be completed executing the critical section before any processor is to continue to the first statement after the "CS end" statement; (v) Iteration statements are delimited by "od" statements and selection statements are delimited by "fi" statements.

Sets are used extensively throughout the parallel algorithms. The set operators that are used by the algorithms include intersection ($\cap$), union ($\cup$), difference ($\setminus$), element of ($\in$), not an element of ($\notin$), subset ($\subseteq$), for all ($\forall$), and set minimum (SetMin). Boolean operators are also used extensively throughout the algorithms. The boolean operators that are used include and ($\wedge$), or ($\vee$), and not ($\neg$). Notation used in the parallel algorithms include **P** for the set of potential aggregate state orderings, **D** for the set of substates of the structure system, n for the number of states in the overall system, p for the number of processors, and $\Sigma = \cup_l \bar{s}(v_l)$.

2.1 Godelization

The parallelization of these algorithms has dictated the need to distribute the states and substates of the system over the number of available processors. This distribution can be done by enumerating the states and substates and distributing them to the appropriate processors using modulo arithmetic. We need an enumeration process by which given a state or substate, we can directly determine the processing element responsible for the storage of and calculation of information for that state or substate and by which given a processing element number, we can directly determine the set of states and substates for which the particular processing element is "responsible". A recommendation is made here for the use of the following enumeration technique which is based on the method for encoding strings called Godel numbering (so named after the logician Kurt Godel) [LEWI 81].

2.1.1 State and Substate Representation

We impose an ordering to the variables of the overall system: $V = \{v_1, v_2, \ldots v_l\}$. Recall that $A = s(v_1) \times s(v_2) \times \cdots \times s(v_l)$ is the set of all potential aggregate states. We impose an ordering on the elements of the sets $s(v_i)$ $(1 \le i \le l)$, and map them ot the set $\bar{s}(v_i) = N_{|s(v_i)|}$.

Let $\mathbf{P} = \bar{s}(v_1) \times \bar{s}(v_2) \times \ldots \times \bar{s}(v_l)$ be the set of all potential aggregate state orderings. Note that $\bar{s}(v_i) \in 1 \le j \le |S|, 0 \in \bar{s}(v_i)\ \forall 1 \le i \le l$ and there is a bijection between $\bar{s}(v_i)$ and $s(v_i)\ \forall 1 \le i \le l$. We can , therefore, represent A for an overall system by **P** as illustrated in Figure 14. We can also represent ${}^0A$, the potential aggregate states of a subsystem, by ${}^0\mathbf{P}$ if we let ${}^0\bar{s}(v_j) = 0\ \forall v_j \notin {}^0V$ as illustrated in Figure 15.

2.1.2 String Enumeration

Consider the potential aggregate state orderings as strings from the alphabet $\Sigma = \cup_i \bar{s}(v_i)$. Each string in **P** (or ${}^0\mathbf{P}$) can be viewed as an integer in base $|\Sigma|$ notation. We define a function gn: ${}^0\mathbf{P} \rightarrow /N$ as follows: If $w = \left(\bar{s}_{i_1}, \bar{s}_{i_2}, \cdots \bar{s}_{i_l}\right)$ where $\bar{s}_{i_j} \in \bar{s}(v_j)$ then $gn(w) = |\Sigma|^{l-1} \bullet \bar{s}_{i_1} + |\Sigma|^{l-2} \bullet \bar{s}_{i_2} + \cdots + |\Sigma|^1 \bullet \bar{s}_{i_{l-1}} + \bar{s}_{i_l}$ [LEWI 81]. We say that gn(w) is the godel number of the string w. Figure 16 illustrates the godel numbers for the states and substates of the example system. Note that gn is one-to-one, but it is not necessarily onto. The function gn is onto only when every variable of the system can take on the same number of values (i.e. $|s(v_i)| = |s(v_j)|$).

v_1	v_2	v_3	potential aggregate state orderings
0	0	0	(1,1,1)
0	0	1	(1,1,2)
0	0	2	(1,1,3)
0	1	0	(1,2,1)
0	1	1	(1,2,2)
0	1	2	(1,2,3)
1	0	0	(2,1,1)
1	0	1	(2,1,2)
1	0	2	(2,1,3)
1	1	0	(2,2,1)
1	1	1	(2,2,2)
1	1	2	(2,2,3)

Figure 14: Potential Aggregate State Orderings of an Overall System

v_2	v_3	potential aggregate state orderings
0	0	(0,1,1)
0	1	(0,1,2)
0	2	(0,1,3)
1	0	(0,2,1)
1	1	(0,2,2)
1	2	(0,2,3)

v_2	v_3	potential aggregate state orderings
0	0	(1,0,1)
0	1	(1,0,2)
0	2	(1,0,3)
1	0	(2,0,1)
1	1	(2,0,2)
1	2	(2,0,3)

v_2	v_3	potential aggregate state orderings
0	0	(1,1,0)
0	1	(1,2,0)
1	0	(2,1,0)
1	1	(2,2,0)

Figure 15: Potential Aggregate State Orderings of Substates

v_1	v_2	v_3	w	Base Conversion	gn(w)
		0	(0, 0, 1)	4^2 (0) + 4(0) + (1)	1
		1	(0, 0, 2)	4^2 (0) + 4(0) + (2)	2
		2	(0, 0, 3)	4^2 (0) + 4(0) + (3)	3
	0		(0, 1, 0)	4^2 (0) + 4(1) + (0)	4
	0	0	(0, 1, 1)	4^2 (0) + 4(1) + (1)	5
	0	1	(0, 1, 2)	4^2 (0) + 4(1) + (2)	6
	0	2	(0, 1, 3)	4^2 (0) + 4(1) + (3)	7
	1		(0, 2, 0)	4^2 (0) + 4(2) + (0)	8
	1	0	(0, 2, 1)	4^2 (0) + 4(2) + (1)	9
	1	1	(0, 2, 2)	4^2 (0) + 4(2) + (2)	10
	1	2	(0, 2, 3)	4^2 (0) + 4(2) + (3)	11
0			(1, 0, 0)	4^2 (1) + 4(0) + (0)	16
0		0	(1, 0, 1)	4^2 (1) + 4(0) + (1)	17
0		1	(1, 0, 2)	4^2 (1) + 4(0) + (2)	18
0		2	(1, 0, 3)	4^2 (1) + 4(0) + (3)	19
0	0		(1, 1, 0)	4^2 (1) + 4(1) + (0)	20
0	0	0	(1, 1, 1)	4^2 (1) + 4(1) + (1)	21
0	0	1	(1, 1, 2)	4^2 (1) + 4(1) + (2)	22
0	0	2	(1, 1, 3)	4^2 (1) + 4(1) + (3)	23
0	1		(1, 2, 0)	4^2 (1) + 4(2) + (0)	24
0	1	0	(1, 2, 1)	4^2 (1) + 4(2) + (1)	25
0	1	1	(1, 2, 2)	4^2 (1) + 4(2) + (2)	26
0	1	2	(1, 2, 3)	4^2 (1) + 4(2) + (3)	27
1			(1, 0, 0)	4^2 (1) + 4(0) + (0)	32
1		0	(2, 0, 1)	4^2 (2) + 4(0) + (1)	33
1		1	(2, 0, 2)	4^2 (2) + 4(0) + (2)	34
1		2	(2, 0, 3)	4^2 (2) + 4(0) + (3)	35
1	0		(2, 1, 0)	4^2 (2) + 4(1) + (0)	36
1	0	0	(2, 1, 1)	4^2 (2) + 4(1) + (1)	37
1	0	1	(2, 1, 2)	4^2 (2) + 4(1) + (2)	38
1	0	2	(2, 1, 3)	4^2 (2) + 4(1) + (3)	39
1	1		(2, 2, 0)	4^2 (2) + 4(2) + (0)	40
1	1	0	(2, 2, 1)	4^2 (2) + 4(2) + (1)	41
1	1	1	(2, 2, 2)	4^2 (2) + 4(2) + (2)	42
1	1	2	(2, 2, 3)	4^2 (2) + 4(2) + (3)	43

Figure 16: Godel Numbering

Using this technique to enumerate each state and substate by a unique decimal number, we can then distribute the computation and storage of information over processors using modulo arithmetic. We now have a technique by which given a system state or substate, the processor handling that state or substate can be easily computed. We also need a technique by which given a processing element number, the states and substates handled by that processor can be easily computed. This can be easily accomplished by a simple base conversion of decimal to $|\Sigma|$. Figure 17 shows the result of inversing the gn function to find the potential aggregate state orderings. Note that each processor is responsible for maintaining and/or calculating the information for the potential aggregate state orderings which have godel numbers ranging between $\left(\text{PE\#} \times \left\lceil \frac{|P|}{p} \right\rceil\right)$ and $\left((\text{PE\#}+1) \times \left\lceil \frac{|P|}{p} \right\rceil\right) - 1$ where **P** is the superset of potential aggregate state orderings for the overall system, and p is the number of processors.

The following routine can be employed to calculate the godel number of a string.

```
(1)   routine gn(word: string)
(2)   num    :   integer
(3)   factor :   integer
(4)   begin routine
(5)       set num = 0
(6)       set factor = 1
(7)       for i = / downto 1 do
```

	$gn^{-1}(0)$	=	(0, 0, 0)		$gn^{-1}(32)$	=	(2, 0, 0)
	$gn^{-1}(1)$	=	(0, 0, 1)		$gn^{-1}(33)$	=	(2, 0, 1)
	$gn^{-1}(2)$	=	(0, 0, 2)		$gn^{-1}(34)$	=	(2, 0, 2)
	$gn^{-1}(3)$	=	(0, 0, 3)		$gn^{-1}(35)$	=	(2, 0, 3)
	$gn^{-1}(4)$	=	(0, 1, 0)		$gn^{-1}(36)$	=	(2, 1, 0)
	$gn^{-1}(5)$	=	(0, 1, 1)		$gn^{-1}(37)$	=	(2, 1, 1)
	$gn^{-1}(6)$	=	(0, 1, 2)		$gn^{-1}(38)$	=	(2, 1, 2)
	$gn^{-1}(7)$	=	(0, 1, 3)		$gn^{-1}(39)$	=	(2, 1, 3)
	$gn^{-1}(8)$	=	(0, 2, 0)		$gn^{-1}(40)$	=	(2, 2, 0)
	$gn^{-1}(9)$	=	(0, 2, 1)		$gn^{-1}(41)$	=	(2, 2, 1)
	$gn^{-1}(10)$	=	(0, 2, 2)		$gn^{-1}(42)$	=	(2, 2, 2)
	$gn^{-1}(11)$	=	(0, 2, 3)		$gn^{-1}(43)$	=	(2, 2, 3)
*	$gn^{-1}(12)$	=	(0, 3, 0)	*	$gn^{-1}(44)$	=	(2, 3, 0)
*	$gn^{-1}(13)$	=	(0, 3, 1)	*	$gn^{-1}(45)$	=	(2, 3, 1)
*	$gn^{-1}(14)$	=	(0, 3, 2)	*	$gn^{-1}(46)$	=	(2, 3, 2)
*	$gn^{-1}(15)$	=	(0, 3, 3)	*	$gn^{-1}(47)$	=	(2, 3, 3)
	$gn^{-1}(16)$	=	(1, 0, 0)	*	$gn^{-1}(48)$	=	(3, 0, 0)
	$gn^{-1}(17)$	=	(1, 0, 1)	*	$gn^{-1}(49)$	=	(3, 0, 1)
	$gn^{-1}(18)$	=	(1, 0, 2)	*	$gn^{-1}(50)$	=	(3, 0, 2)
	$gn^{-1}(19)$	=	(1, 0, 3)	*	$gn^{-1}(51)$	=	(3, 0, 3)
	$gn^{-1}(20)$	=	(1, 1, 0)	*	$gn^{-1}(52)$	=	(3, 1, 0)
	$gn^{-1}(21)$	=	(1, 1, 1)	*	$gn^{-1}(53)$	=	(3, 1, 1)
	$gn^{-1}(22)$	=	(1, 1, 2)	*	$gn^{-1}(54)$	=	(3, 1, 2)
	$gn^{-1}(23)$	=	(1, 1, 3)	*	$gn^{-1}(55)$	=	(3, 1, 3)
	$gn^{-1}(24)$	=	(1, 2, 0)	*	$gn^{-1}(56)$	=	(3, 2, 0)
	$gn^{-1}(25)$	=	(1, 2, 1)	*	$gn^{-1}(57)$	=	(3, 2, 1)
	$gn^{-1}(26)$	=	(1, 2, 2)	*	$gn^{-1}(58)$	=	(3, 2, 2)
	$gn^{-1}(27)$	=	(1, 2, 3)	*	$gn^{-1}(59)$	=	(3, 2, 3)
*	$gn^{-1}(28)$	=	(1, 3, 0)	*	$gn^{-1}(60)$	=	(3, 3, 0)
*	$gn^{-1}(29)$	=	(1, 3, 1)	*	$gn^{-1}(61)$	=	(3, 3, 1)
*	$gn^{-1}(30)$	=	(1, 3, 2)	*	$gn^{-1}(62)$	=	(3, 3, 2)
*	$gn^{-1}(31)$	=	(1, 3, 3)	*	$gn^{-1}(63)$	=	(3, 3, 3)

The (*) indicates strings not found in the language due to the gn function not being onto.

Figure 17: Inverse of the Godel Numbering Function

(8) set num = num + (word[i] × factor)

(9) set factor = factor ×|Σ|

(10) od

(11) return (num)

(12) end routine

The following routine can be employed to calculate the inverse godel number of an integer.

(1) routine gn^{-1} (num: integer)

(2) word : string

(3) begin routine

(4) for i = *l* downto 1 do

(5) set word[i] = num modulo |Σ|

(6) set num = $\left\lfloor \frac{num}{|\Sigma|} \right\rfloor$

(7) od

(8) return (word)

(9) end routine

The structural complexity of the gn routine is O(*l*), and the structural complexity of the gn^{-1} routine is O(*l*) where *l* is the number variables in the overall system. A description of the structural complexity evaluation can be found in appendix C.

2.2 Algorithm Initialization

The reconstructability analysis algorithms parallelized here take as input a structure system. This structure system may be input in a variety of ways depending on the application being analyzed. This structure system must, however, be formatted for use by these algorithms.

Prior to local memory allocation, the determination of the variables of the structure system must be made and an ordering must be imposed on the variables. Let l be the number of variables in the structure system. The variables are then ordered by a one-to-one correspondence between the variables and the integers between 1 and l, inclusively. A determination must also be made of the set $S = \cup_l s(v_l)$ of all possible values of variables. An ordering is also imposed on the elements of S : $S = \{s_j | s_j \in \cup_l s(v_l), 1 \leq j \leq |S|\}$.

Once these parameters have been determined, local memory may be allocated. Let **P** (where $|\mathbf{P}| = \Sigma^{|l|}$) be the universe set of the potential aggregate state orderings (as described in section 2.1) for the given structure system. Each PE is allocated two arrays indexed between 0 and $\left\lceil \frac{|\mathbf{P}|}{p} \right\rceil - 1$. One array is labeled state. If $\mathrm{gn}^{-1}\left(\left(\mathrm{PE\#} \times \left\lceil \frac{|\mathbf{P}|}{p} \right\rceil\right) + \mathrm{index}\right)$ is the potential aggregate state ordering that corresponds to a state of the overall system then the state array contains a value of 1, if it corresponds to a substate of the overall system, then the state array contains a value of 0, and if it corresponds to neither (due to the function gn not being onto), then the

state array contains a value of -1. A second array is labeled *f* which contains the known values of the behavior function.

The following algorithm is employed to initialize the two arrays in local memory for use by the parallelized Reconstructability Analysis algorithms. The local memory for the algorithm initialization is given in figure 18.

```
                                        /* initialize state variable to 1, 0,
                                        or -1 */
(1)   {∀[0 ≤ i < p]}
(2)    par begin
(3)       for j = 0 to (⌈|P|/p⌉ − 1) do
(4)          set PE(i).word = gn^-1 ((⌈|P|/p⌉ × i) + j)
(5)          set PE(i).state[j] = 1
(6)          for k = 1 to l do
(7)             if (PE(i).word[k] = 0) then
(8)                set PE(i).state[j] = -1
(9)             fi
(10)         od
(11)      od
(12)   par end
                                        /* input substate information to
                                        initialize behavior function
```

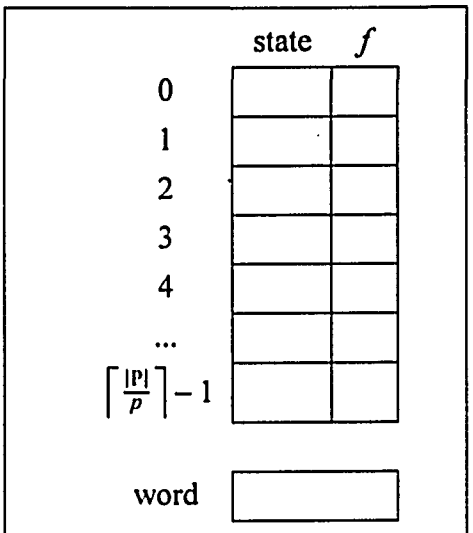


Figure 18: Local Memory for Algorithm Initialization

values */

(13) while (more substate information remains to be input) do

(14) for i = 1 to l do

(15) get input for the i^{th} variable

(16) if (no value is given for the i^{th} variable) then

(17) set word[i] = 0

(18) else

(19) set word[i] = position of the value in the ordering of variable values

(20) fi

(21) od

(22) set $PE\left(\left\lfloor \frac{gn(word)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right)$ .state[gn(word) modulo $\lceil \frac{|P|}{p} \rceil$]

(23) input $PE\left(\left\lfloor \frac{gn(word)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right)$ .f[gn(word)modulo $\lceil \frac{|P|}{p} \rceil$]

(24) od

The structural complexity of the algorithm initialization is $O(|D|^l|\Sigma|)$. A description of the structural complexity evaluation can be found in appendix C.

2.3 Parallelization of the Determination of Unbiased Reconstructions

2.3.1 Generating Independent Substates in Parallel

The first step in the determination of unbiased reconstructions is the generation of independent substates. The algorithm described in this section generates the set E

selecting exactly one element from each equivalence class of null extensions as described in section 2.1.

This set generation is performed in parallel by having each processing element (PE) test all of its state variables to determine whether that index corresponds to a godel number that maps to a substate of the overall system. If such a mapping exists, the null extension is determined by changing each potential aggregate ordering value of zero to one. If the null extension is not already present in the set of null extensions of the set E, then the substate is added to the set E.

The local memory needed to generate independent substates is illustrated in figure 19, and the global memory is illustrated in figure 20.

/* initialize global set */

(1) set NullExtension = $\varnothing$

(2) $\{\forall[1 \le i < p]\}$

(3) par begin

(4) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil - 1$ do

/* if the gn maps to a substate then */

(5) set PE(i).E[j] = 0

(6) if (PE(i).state[j] = 0) then

(7) set PE(i).word = $gn^{-1}\left(\left(i \times \left\lceil \frac{|P|}{p} \right\rceil\right) + j\right)$

/* determine the null extension of

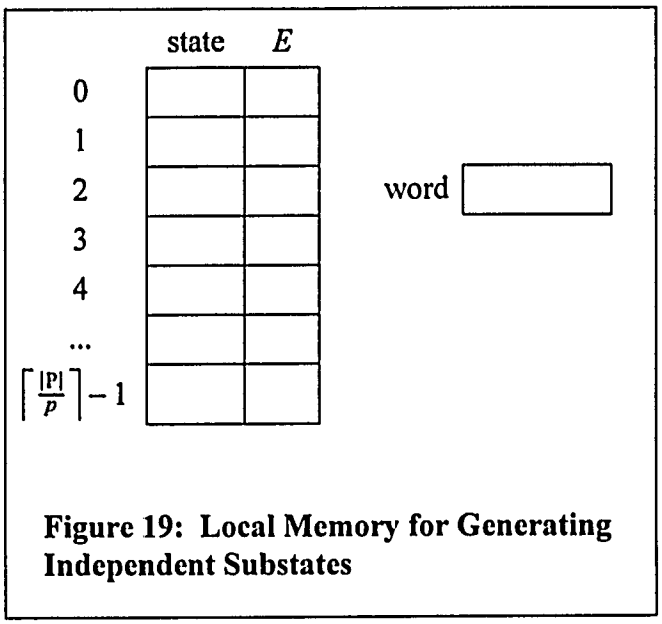


Figure 19: Local Memory for Generating Independent Substates

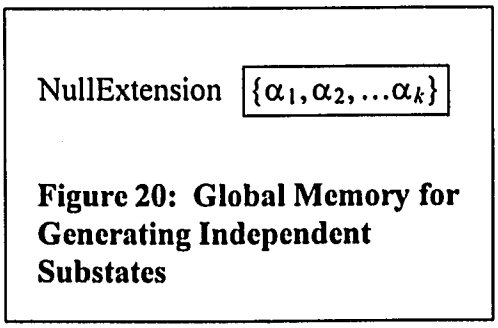


Figure 20: Global Memory for Generating Independent Substates

```
                                                    the substate */
(8)             for k = 1 to l do
(9)                 if (PE(i).word[k] = 0) then
(10)                    set PE(i).word[k] = 1
(11)                fi
(12)            od
                                                    /* if the null extension is not the
                                                    null extension of a substate
                                                    already in the set E then */
(13)            begin CS
(14)                if (PE(i).word ∉ NullExtension) then
(15)                    set NullExtension = NullExtension ∪ {PE(i).word}
                                                    /* add the substate to the set E */
(16)                    set PE(i).E[j] = 1
(17)                fi
(18)            end CS
(19)        fi
(20)    od
(21)  par end
```

The structural complexity of generating independent substates is $O(\frac{|P|}{p}n)$. A description of the structural complexity evaluation is located in appendix C.

2.3.2 Partitioning the States into Disjoint Sets in Parallel

The second step in the determination of unbiased reconstructions is the partitioning the states into disjoint sets. The criteria for the partitioning is that no two elements of the partition have mutual superstates. The substates of the structure system are partitioned into sets C which are distributed over the p processors.

This algorithm uses a global array of partition superstates. All of the partitions are constructed simultaneously by inserting the superstates of the substates of the iteration into the first partition that does not contain any of its superstates. The set of superstates for that partition is unioned with the set of superstates for the newly inserted substate. Upon each iteration, the superstates of the substates of that iteration are calculated for use in this partitioning and the set is saved in local memory for use in calculating the equations of the reconstruction hypothesis. As the algorithm iterates, it also calculates the value of a for the extra equation of the reconstruction hypothesis for each partition. Once iteration is complete. The extra equation of the reconstruction hypothesis for each partition is constructed.

The local memory needed for the partitioning the states into disjoints sets is illustrated in figure 21, and the global memory needed is illustrated in figure 22.

/* initialize variables */

(1) for i = 1 to $|E|$ do

(2) set extraa[i] = 1

(3) set CSuperstates[i] = $\varnothing$

(4) od

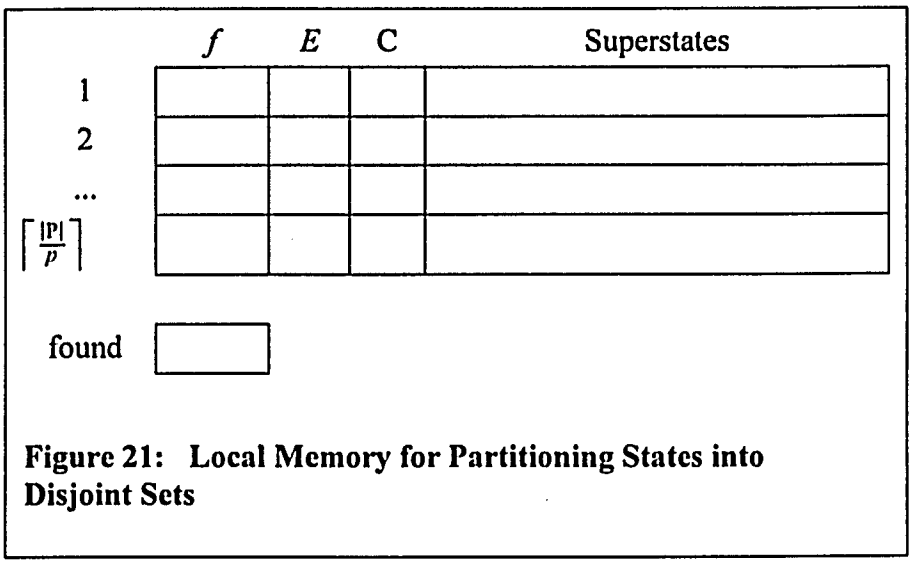


Figure 21: Local Memory for Partitioning States into Disjoint Sets

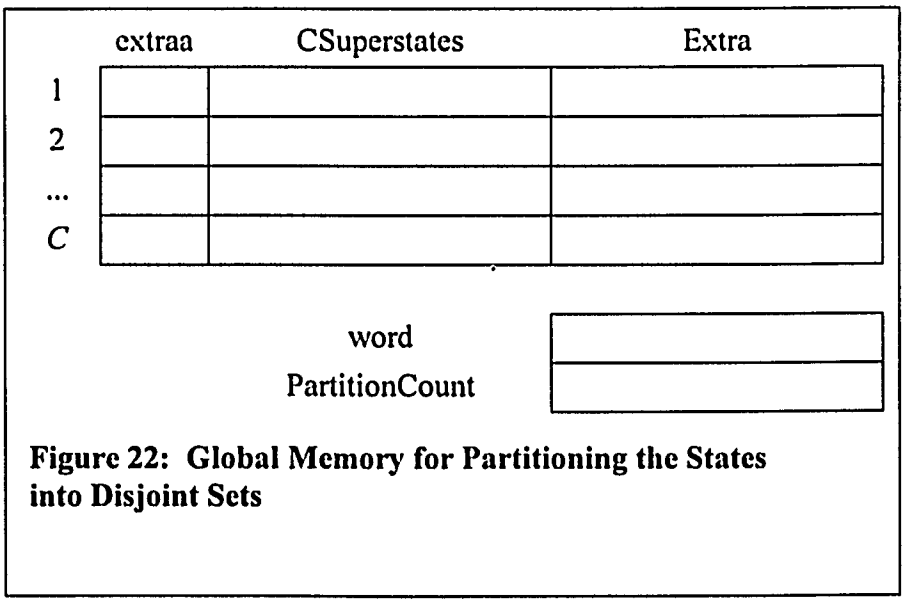


Figure 22: Global Memory for Partitioning the States into Disjoint Sets

(5) set PartitionCount = 0

(6) $\{\forall[0 \le i < p]\}$

(7) par begin

(8) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil - 1$ do

(9) set PE(i).C[j] = 0

(10) if (PE(i).E[j] = 1) then

(11) call GenerateSuperstates (gn^{-1} ($\left(i \times \left\lceil \frac{|P|}{p} \right\rceil\right) + j$),PE(i).Superstates[j])

/* find partition to place state */

(12) set PE(i).C[j] = 1

(13) set PE(i).found = false

(14) CS begin

(15) while ((not PE(i).found) do

(16) if ((PE(i).Superstates[j] $\cap$ CSuperstates[PE(i).C[j]])

$= \varnothing$) then

(17) set PE(i).found = true

(18) else

(19) set PE(i).C[j] = PE(i).C[j] + 1

(20) fi

(21) od

(22) set CSuperstates[PE(i).C[j]] =

CSuperstates[PE(i).C[j]] $\cup$ PE(i).Superstates[j]

```
(23)            CS end
                                        /* created new partition */
(24)            if (PE(i).C[j] > PartitionCount) then
(25)                set PartitionCount = PE(i).C[j]
(26)            fi
                                        /* calculate a for extra equation */
(27)            set extraa[PE(i).C[j]] = extraa[PE(i).C[j]] - PE(i).f[j]
(28)        fi
(29)    od
(30)    par end
                                        /* construct extra equation in
                                        partition */
(31)    for i = 1 to l do
(32)        set word[i] = 0
(33)    od
(34)    GenerateSuperstates(word, Extra[1])
(35)    for i = 2 to PartitionCount do
(36)        set Extra[i] = Extra[1] \ CSupersets[i]
(37)    od
(38)    set Extra[1] = Extra[1] \ CSupersets[1]
```

The structural complexity of this partitioning process is O(E^2n). A description of the structural complexity evaluation can be found in appendix C.

The parallel algorithm for partitioning states into disjoint sets calls the recursive routine GenerateSuperstates. This routine is described below. The memory needed by this routine is illustrated in figure 23. The structural complexity of this routine is taken to be $O(|\Sigma|^{l})$. A justification of this structural complexity is given in appendix C.

```
(1)   routine GenerateSuperstates(word: string, {ref} Superset: set of strings)
(2)       routine generate(word : string, position : integer, {ref} S : set of strings)
(3)       begin routine
(4)           if (Word[Position] = 0) then
(5)               for k = 1 to |Σ| – 1 do
(6)                   set Word[Position] = k
(7)                   if (PE(⌊ gn(word) / ⌈|P|/p⌉ ⌋).state[(gn(word) modulo ⌈|P|/p⌉]) > 0) then
(8)                       set S = S ∪ {Word}
(9)                   else
(10)                      for m = position + 1  to l do
(11)                          call generate(word, m, S)
(12)                      od
(13)                  fi
(14)              od
(15)          fi
(16)      end routine
```

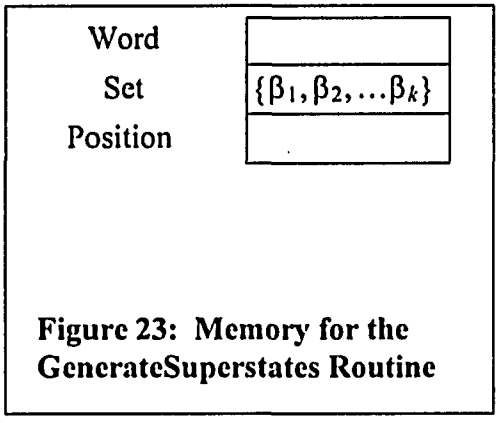


Figure 23: Memory for the GenerateSuperstates Routine

(17) begin routine

(18) call generate(word, 1, Superset)

(19) end routine

2.3.3 Parallelization of the Reconstruction Process

The third step in the determination of unbiased reconstructions is the reconstruction process. The partition of the substates into disjoint sets is used as the reconstruction hypothesis for the reconstruction process.

The reconstruction process is performed in parallel by first initializing $\hat{f}$ to a flat distribution. Until convergence is obtained, the value of $\hat{a}$ is calculated (including the extra equation of the partition) and then a new value of $\hat{f}$ is obtained (including the extra equation of the partition) by finding the product old$\hat{f} \times \frac{a_j}{\hat{a}_j}$ for each partition. While $\hat{a}$ is being calculated in parallel, the superstates are tagged as to which substate is using its $\hat{f}$ value in the current partition. All of the remaining superstates are tagged with a -1 indicating that they are part of the extra equation for the current partition. This allows the calculation of the new $\hat{f}$ value in parallel. Once all of the reconstruction hypothesis have been calculated (that is once the algorithm has iterated through all of the partitions), convergence is tested.

The local memory needed for the reconstruction process is illustrated in figure 24, and the global memory needed is illustrated in figure 25.

/* initialize $\hat{f}$ to a flat

distribution */

(1) {$\forall[0 \le i < p]$}

(2) par begin

(3) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil - 1$ do

(4) if (PE(i).state[j] = 1) then

(5) set PE(i).old$\hat{f}$ [j] = $\frac{1}{n}$

(6) fi

(7) od

(8) par end

(9) set converged = false

(10) while (not converged) do

(11) for PartitionNum = 1 to PartitionCount do

/* calculate $\hat{a}$ */

(12) {$\forall[0 \le i < p]$}

(13) par begin

(14) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil$ -1 do

(15) if (PE(i).C[j] = PartitionNum) then

(16) set PE(i).$\hat{a}$ [j] = 0

(17) set PE(i).TempSuper = PE(i).Superstates[j]

(18) while (PE(i).TempSuper $\neq \varnothing$) do

(19) set PE(i).element = SetMin(PE(i).TempSuper)

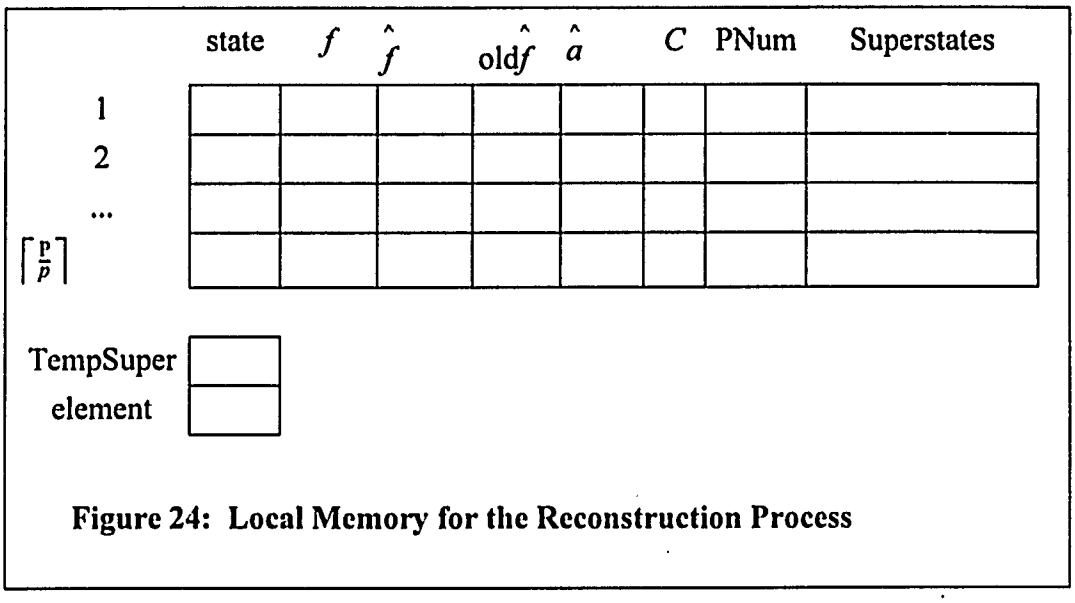


Figure 24: Local Memory for the Reconstruction Process

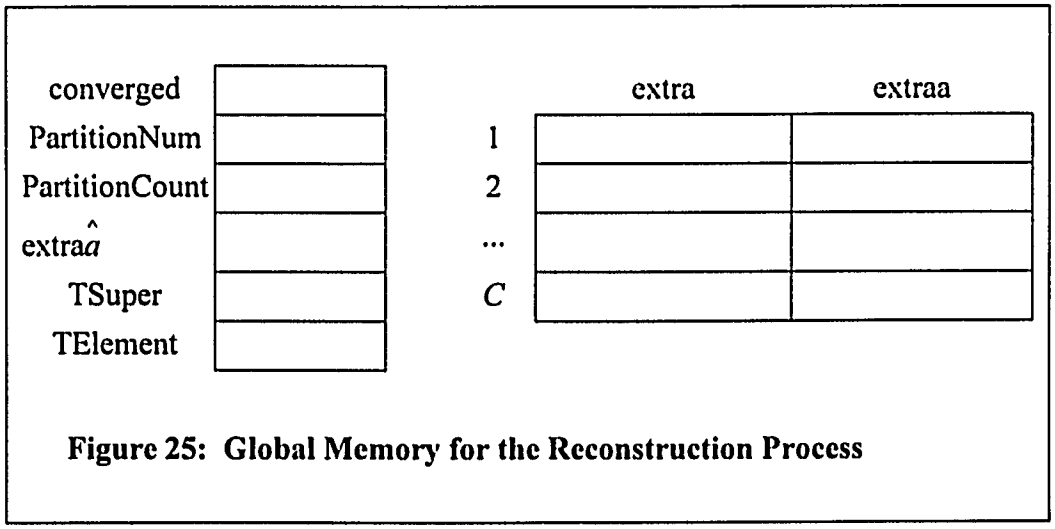


Figure 25: Global Memory for the Reconstruction Process

(20) set PE(i).TempSuper = PE(i).TempSuper \

{PE(i)..element}

(21) set PE(i).$\hat{a}$ [j] = PE(i).$\hat{a}$ [j] +

PE($\left\lfloor \frac{\text{gn(PE}(i)\text{.element)}}{\lceil \frac{p}{p} \rceil} \right\rfloor$).old$\hat{f}$ [gn(PE(i).element) modulo

$\lceil \frac{|P|}{p} \rceil$]

/* designate substate for this

partition */

(22) set PE($\left\lfloor \frac{\text{gn(PE}(i)\text{.element)}}{\lceil \frac{p}{p} \rceil} \right\rfloor$).PNum[gn(PE($i$).element)

modulo$\lceil \frac{|P|}{p} \rceil$) = $\left(i \times \lceil \frac{|P|}{p} \rceil\right) + j$

(23) od

(24) fi

(25) od

(26) par end

/* calculate extra $\hat{a}$ */

(27) set extra$\hat{a}$ = 0

(28) set TSuper = CSupersets[PartitionNum]

(29) while (TSuper ≠ ∅) do

(30) set TElement = SetMin(TSuper)

(31) set TSuper = TSuper \ {TElement}

(32) set extra$\hat{a}$ = extra$\hat{a}$ +

$$PE\left(\left\lfloor \frac{gn(TElement)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).old\hat{f}\,[gn(TElement) \text{ modulo } \lceil \tfrac{|P|}{p} \rceil$$

(33) $$\text{set } PE\left(\left\lfloor \frac{gn(TElement)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).PNum[gn(TElement) \text{ modulo } \lceil \tfrac{|P|}{p} \rceil\,] = -1$$

(34) od

$$/*\ \text{set } \hat{f} = old\hat{f} \times \frac{a_j}{\hat{a}_j}\ */$$

(35) $\{\forall[0 \le i < p]\}$

(36) par begin

(37) for j = 0 to $\lceil \frac{|P|}{p} \rceil$ -1 do

(38) if (PE(i).state[j] = 1) then

(39) if (PE(i).PNum[j] = -1) then

(40) $$\text{set } PE(i).\hat{f}\,[j] = PE(i).old\hat{f}\,[j] \times \frac{extraa[PartitionNum]}{extra\,\hat{a}}$$

(41) else

(42) $$\text{set } PE(i).\hat{f}\,[j] = PE(i).old\hat{f}\,[j] \times \frac{PE\left(\left\lfloor \frac{PE(i).PNum}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).\hat{f}\left[PE(i).PNum[j] \text{ modulo } \lceil \frac{|P|}{p} \rceil\right]}{PE\left(\left\lfloor \frac{PE(i).PNum}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).\hat{a}\left[PE(i).PNum[j] \text{ modulo } \lceil \frac{p}{p} \rceil\right]}$$

(43) fi

(44) fi

(45) od

(46) par end

(47) od

/* test convergence */

(48) set converged = true

(49) $\{\forall[0 \le i < p]\}$

(50) par begin

(51) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil - 1$ do

(52) if PE(i).state[j] = 1 then

(53) set converged = converged $\wedge$

$$\left(\left| \text{PE}(i).\hat{f}[\text{j}] - \text{PE}(i).\text{old}\hat{f}[\text{j}] \right| < \varepsilon \right)$$

(54) set PE(i).old$\hat{f}$ [j] = PE(i).$\hat{f}$ [j]

(55) fi

(56) od

(57) par end

(58) od

The structural complexity of the reconstruction process is $O(\delta_U C \frac{|P|}{p} n)$ where δ_U is the rate of convergence of the reconstruction process and C is the number of optimal partitions of states into disjoint sets. A description of the structural complexity evaluation can be found in appendix C.

2.4 Parallelization of a Greedy Algorithm for a Generalization of the Reconstruction Problem

The greedy algorithm for the generalization of the reconstruction problem takes as input the local memory initialized as described in section 2.2. Prior to

iteration, however, a set E must be constructed in global memory of states of the structure system (or substates of the overall system) that are known. Recall that these substates need not define the overall system completely to proceed. Once this set is constructed, the behavior function of the overall system is initialized to the uniform distribution. Until convergence is obtained, the substate is selected from the set E to add to the set D which maximizes the choice function, and the unbiased reconstruction is performed on the updated version of set D.

The maximum value of the choice function is found by distributing the values of the set D and the value of its choice function over the half of the processors that are numbered the highest (excluding processor p - 1). The processors are then treated as an array implementation of a binary tree. At each step, the next highest level (the parent of two nodes in the binary tree) retrieve the largest of the given level (its two children). Until all element of the set D have been evaluated, at each step, the highest value is "rolling" up the tree while new values from D are being placed into the root level of the binary tree. When the "rolling" of the numbers is complete, PE(1) contains the value of the largest choice function value and its substate.

The local memory needed for the greedy algorithm is illustrated in figure 26, and the global memory needed is illustrated in figure 27.

Let k be the number of elements in the set E (i.e. $k = |E|$).

/* construct set E */

(1) set $E = \varnothing$

(2) $\{\forall [0 \leq i < p]\}$

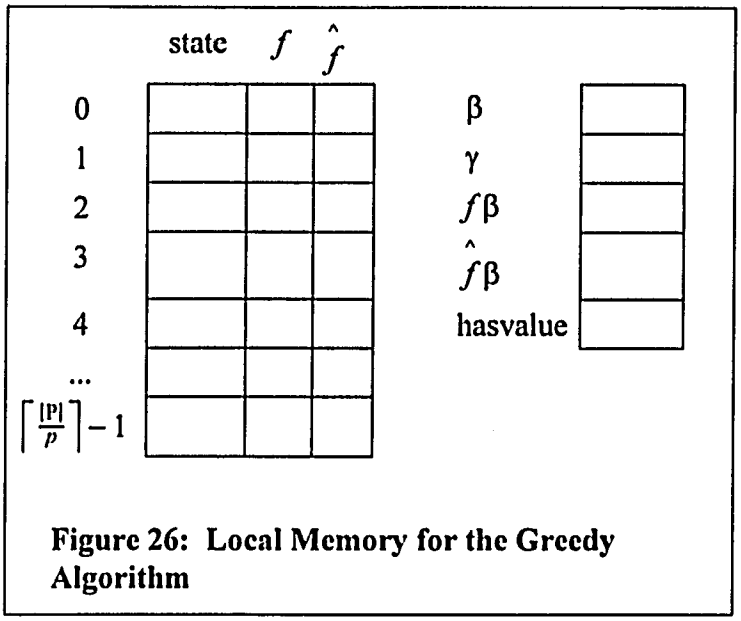


Figure 26: Local Memory for the Greedy Algorithm

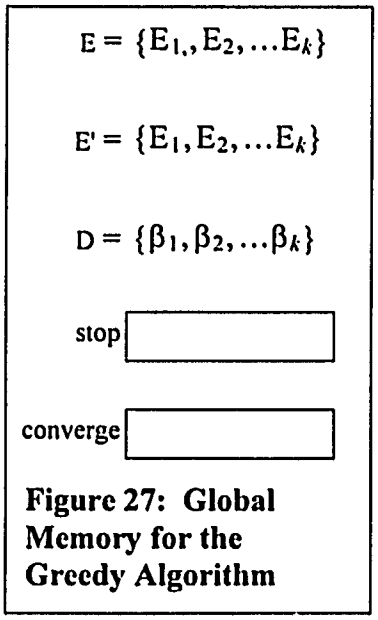


Figure 27: Global Memory for the Greedy Algorithm

(3) par begin

(4) for j = 0 to $\left\lceil \frac{|P|}{p} \right\rceil - 1$ do

(5) if (PE(i).state[j] = 0 then

(6) set $E = E \cup \text{gn}^{-1}\left(\left(i \times \left\lceil \frac{|P|}{p} \right\rceil\right) + \text{j}\right)$

(7) fi

(8) od

(9) par end

/* initialize behavior function approximation to the uniform distribution */

(10) $\{\forall[0 \le i < p]\}$

(11) par begin

(12) for j = 0 to $\left(\left\lceil \frac{|P|}{p} \right\rceil - 1\right)$ do

(13) if PE(i).State[j] = 1 then

(14) set PE(i).$\hat{f}$[j] = $\frac{1}{n}$

(15) fi

(16) od

(17) par end

(18) set $D = \varnothing$

(19) set stop = ($|D|$ = sizelimit)

(20) set converge = false

(21) while (($\neg$ stop) $\wedge$ ($\neg$ converge)) do

/* select one $\beta \in E$ to add to D

such that $\beta = \arg\max_{\beta \in E} \gamma(\beta)$ */

(22) set E' = E

(23) while (|E'| > 0) do

(24) set j = $\frac{p}{2} - 1$

(25) while ((|E'| > 0) $\wedge$ (j $\leq p$ - 2)) do

(26) set PE(j).β=some e$\in$ E'

(27) od

(28) set E' = E' \ {e}

(29) $\left\{\forall\left[\left(\frac{p}{2}-1\right) \leq i < (p-1)\right]\right\}$

/* retrieve f and $\hat{f}$ values */

(30) par begin

(31) set $PE(i).f\beta = PE\left(\left\lfloor \frac{gn(PE(j).\beta)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).f\left[gn(\beta)\text{modulo} \lceil \frac{|P|}{p} \rceil\right]$

(32) set $PE(i).\hat{f}\beta = PE\left(\left\lfloor \frac{gn(PE(j).\beta)}{\lceil \frac{|P|}{p} \rceil} \right\rfloor\right).\hat{f}\left[gn(\beta)\text{modulo} \lceil \frac{|P|}{p} \rceil\right]$

(33) set $PE(i).\gamma = PE(i).f\beta \log_2 \frac{PE(i).f\beta}{PE(i).\hat{f}\beta} + (1 + PE(i).f\beta)\log_2 \frac{(1-PE(i).f\beta)}{(1-PE(i).\hat{f}\beta)}$

(34) par end

(35) $\{\forall[\left(0 \leq i < \frac{p}{2} - 1\right) \wedge (\neg PE(i).\text{hasvalue}) \wedge$

$(PE(2i).\text{hasvalue}) \wedge (PE(2i+1).\text{hasvalue})]\}$

(36) par begin

(37) if (PE($2i$).γ > PE($2i + 1$).γ) then

(38) set PE(i).γ = PE($2i$).γ

(39) set PE(i).β = PE($2i$).β

(40) else

(41) set PE(i).γ = PE($2i + 1$).γ

(42) set PE(i).β = PE($2i + 1$).β

(43) fi

(44) set PE(i).hasvalue = true

(45) set PE($2i$).hasvalue = false

(46) set PE($2i$+1).hasvalue = false

(47) par end

(48) $\{\forall[(0 \leq i < p - 1) \wedge (\neg$PE($i$).hasvalue$) \wedge$
(PE($2i$).hasvalue$) \wedge (\neg$PE($2i + 1$).hasvalue$)]\}$

(49) par begin

(50) set PE(i).γ = PE($2i$).γ

(51) set PE(i).β = PE($2i$).β

(52) par end

(53) $\{(i = 0) \wedge$ (PE(0).hasvalue$) \wedge$
(PE(1).hasvalue$) \wedge$ (PE(2).hasvalue)$\}$

(54) par begin

(55) if ((PE(1).γ > PE(2).γ) ∧ (PE(1).γ > PE(0).γ))then

(56) set PE(0).γ = PE(1).γ

(57) set PE(0).β = PE(1).β

(58) else

(59) if ((PE(2).γ > PE(1).γ) (PE(2).γ > PE(0).γ))then

(60) set PE(0).γ = PE(2).γ

(61) set PE(0).β = PE(2).β

(62) fi

(63) fi

(64) set PE(1).hasvalue = false

(65) set PE(2).hasvalue = false

(66) par end

(67) {∀[0 ≤ i < p − 1]}

(68) set continue = (continue ∨ PE(i).hasvalue)

(69) od

(70) set D = D ∪{PE(0).β}

(71) set E = E \ {PE(0).β}

(72) U(D) /* Compute the unbiased reconstruction on this new D. Note the unbiased reconstruction changes the $\hat{f}$ values for the set D. */

/* test convergence */

(73) set converge = true

(74) $\{\forall[0 \le i < p]\}$

(75) par begin

(76) for $j = 0$ to $\left(\left\lceil \frac{|P|}{p} \right\rceil - 1\right)$ do

(77) set converge = converge $\wedge$ ($|PE(i).f[j] - PE(i).\hat{f}[j]| < \varepsilon$)

(78) od

(79) par end

(80) od

Note that the the unbiased reconstruction is performed at each iteration. That is, whenever a new β is added to the set D, the unbiased reconstruction is performed on the new D set. Some modifications must be made to the reconstruction process to perform the unbiased reconstruction on a subset of the substates of the system. First, rather than initializing old$\hat{f}$ to $\frac{1}{n}$, old$\hat{f}$ is initialized to $\hat{f}$ which is simply the value from the previous unbiased reconstruction. Since the greedy algorithm initializes all $\hat{f}$ to $\frac{1}{n}$, the new β in the set D will have the value $\frac{1}{n}$. Second, the reconstruction is performed only on those elements of the set D, and not all substates of the structure system.

The structural complexity of the greedy algorithm is $O(\delta_G \delta_U C \frac{|P|}{p} n)$ where δ_U is the rate of convergence of the reconstruction process and δ_G is the rate of convergence of the greedy algorithm for a generalization of the reconstruction

problem. A description of the structural complexity evaluation can be found in appendix C.

Chapter 3

Conclusion

3.1 Significance of Reconstructability Analysis

One of the more promising contributions of reconstructability analysis is its utilization in lieu of classical multivariate systems. Jones shows that *K*-systems analysis can solve classical multivariate analysis problems more effectively that other existing methods [JONE 86]. Reconstructability analysis does not assume an underlying mathematical structure (as opposed to a multivariate regression method) but rather constructs a system based only on the information given. Furthermore, "*K*-systems theory delivers a surprising and important insight: it is generally incorrect to use any model where effects and interactions are represented statically over subsets of the equations (as in the analysis of variance). Not only is the *K*-system the correct model, but any other model assumed by the user is almost certainly wrong" [JONE 86]. Clearly, assuming an incorrect model can lead to the verification of interactions that do not exist or to the failure to verify interactions that do exist. Reconstructability analysis eliminates this error potential by constructing the "true" model of the system.

K-systems analysis also provides us with the tools necessary to:

"1) Measure the cognitive content of a substate about the overall system.
2) Determine a minimal set of substates to reproduce the system to a desired approximation.
3) Reproduce a unique system from any set of substates, which contains only the knowledge of the substates." [JONE 86]

K-system analysis is a computationally effective theory, not just a theoretical result [JONE 86].

Since only maximum entropy methodologies are envoked, Reconstructability analysis introduces no extraneous information. The transformation of general functions to probabilistic functions does not jeopardize the system structure: "There is no information present in the *k*-system which is not present in the *g*-system" [JONE 85c]. Cavallo and Klir justify the use of maximum entropy with three diverse arguments:

> "1) The maximum entropy probability distribution is the only unbiased distribution, i.e. the only distribution which takes into account all available information (constraints) but no additional (unsupported) assumptions (biases).
> 2) The maximum entropy probability distribution is the most likely distribution. Given a reconstruction hypothesis, each element of the reconstruction family of that hypothesis could have been generated by any number of actual data sets. The largest number of possible data sets which are mutually comparable and compatible with the given reconstruction hypothesis are those which are also compatible with the maximum entropy overall probability distribution.
> 3) Maximizing any function but entropy leads to inconsistencies unless that function has the same maxima as entropy" [CAVA 81b].

3.2 Significance of Parallelization of Reconstructability Analysis Algorithms

Many papers have been written applying reconstructability analysis and maximum entropy methods to various disciplines and sub-disciplines. Some of these applications include software engineering, automated rule learning [TRIVE 93], waveform analysis [COIF 92], social sciences [KLIR 86], urban planning [KUMA 89], memory management [LEWI 59], and many application in the physical sciences

[COIF 92][ERIC 88][BUCK 91][LEVI 78]. The author is also working on applying reconstructability analysis to information retrieval. Clearly, the implementation of the algorithms is imperative to the effective utilization of reconstructability analysis by these disciplines and sub-disciplines.

Jones' algorithms are the first step toward providing scientists with power of reconstructability analysis. Clearly, implementation is the second step. However, since these algorithms grow exponentially in both time and space, sequential implementation does not make this step for many applications. The time complexity is dominated by an exponential growth which is dictated by the recursive nature of the algorithms, and the number of states involved in the search space of the greedy algorithm grows exponentially. Jones has implemented the algorithms sequentially and makes the implementation available in a software package for IBM and IBM compatible microcomputers [JONE 89]. Because the algorithms are so computationally intensive, the software package must be run on a machine with a math co-processor, and the program is limited to ten variables. This variable limitation, alone, makes the use of the implementation infeasible for most applications.

The scientific community was left with a chasm between theoretical application and feasible implementation. For most applications, this gap is as unbreachable today as it was at the inception of reconstructability analysis by George Klir. Klir recognized this problem back in 1981. In the special issue of International Journal of General Systems, Klir lists several areas of further research in reconstructability analysis, including the design of more efficient algorithms

including "large-scale hardware-based parallel processing" [CAVA 81b]. Until now, this need for efficient algorithms has remained unmet.

Figure 28 demonstrates the improvement of the parallel algorithms over the sequential algorithms on the example structure system.

3.3 Final Remarks

Reconstructability analysis is a powerful tool with may applications in many disciplines and sub-disciplines. However, utilization of reconstructability analysis is infeasible in most applications due to the exponential growth of the algorithms in both time and space. Clearly, there is a tremendous need to provide scientists with accessibility to the power of reconstructability analysis. The next step has now been taken in bridging the gap between theoretical application and feasible implementation by the parallelization of the sequential algorithms developed by Jones: the determination of unbiased reconstruction and the greedy algorithm for a generalization of the reconstruction problem.

Since this parallelization does not assume an architectural model, these parallel algorithms are universal to all multiple data architectures. The situation of having more data items in the overall system than processors in the hardware system is addressed by these algorithms. Therefore, implementation of the algorithms involves simply incorporating the communication protocols between processors for the particular architecture employed.

Now that this step has been accomplished, the next step is implementation of the parallel algorithms on particular multiple data architectures. With the completion

Algorithm	Sequential Time	Parallel Time
Generate Independent Substates	$O(\|\mathbf{D}\|^2 l)$ $O(768)$	$O(\|\mathbf{D}\| n)$ $O(192)$
Partition States into Disjoint Sets	$O(C\|E\|^2 n)$ $O(2916)$	$O(\|E\|^2 n)$ $O(972)$
Reconstruction Process	$O(\delta_U Cn^2)$ $O(\delta_U 432)$	$O(\delta_U C\frac{\|P\|}{p} n)$ $O(\delta_U 288)$
Greedy Algorithm for a Generatlization of the Reconstruction Process	$O(\delta_G \delta_U Cn^2)$ $O(\delta_G \delta_U 432)$	$O(\delta_G \delta_U C\frac{\|P\|}{p} n)$ $O(\delta_G \delta_U 288)$

Figure 28: Comparison of Sequential and Parallel Times

of the next step, accesiblilty to the power of reconstructability analysis will finally be available to researchers. Note that further research is needed in the analysis of the rate of convergence for the reconstruction process and in the analysis of the rate of convergence for the greedy algorithm for a generalization of the reconstruction process. The author also intends to explore the advantages and disadvantages to incorporating dynamic memory allocation (perhaps in the form of a binary search tree) over the direct memory access of the hash table used here.

Bibliography

[ADKI 92] Adkins, William A. and Weintraub, Steven H. (1992). "Algebra. An Approach via Module Theory", Springer-Verlag, New York.

[AHO 74] Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. (1974). "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Company, Reading, Massachusetts.

[AHO 83] Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. (1983). "Data Structures and Algorithms", Addison-Wesley Publishing Company, Reading, Massachusetts.

[ALTM 90] Altman, Tom and Chlebus, Bogdan S. (1990). Sorting Roughly Sorted Sequences in Parallel. *Information Processing Letters*, **33**, 297-300.

[BALC 88] Balcazar, Jose Luis, Diaz, Josep, and Gabarro, Joaquim (1988). "Structural Complexity I", Springer-Verlag, Berlin.

[BALD 94] Baldoni, R. and Ciciani, B (1994). Distributed algorithms for multiple entries to a critical section with priority. *Information Processing Letters*, **50**, 165-172.

[BOGA 88] Bogart, Kenneth (1988). "Discrete Mathematics", D. C. Heath Company, Lexington, Massachusetts.

[BRAS 88] Brassand, Gilles, and Bratley, Paul (1988). "Algorithmics. Theory and Practice", Printice Hall, Englewood Cliffs, New Jersey.

[BROE 81] Broekstra, Gerrit (1981). *C*-Analysis of *C*-Structures:. Representation and Evaluation of Reconstruction Hypothesis by Information Measures. *International Journal of General Systems*, **7**, 33-61.

[BROW 59] Brown, David T. (1959). A Note on Approximations to Discrete Probability Distributions. *Information and Control*, **2**, 386-392.

[BUCK 91] Buck, Brian and Macaulay, Vincent A., ed. (1991). "Maximum Entropy in Action." Oxford University Press. New York, New York.

[CARL 91] Carlsson, Svante and Chen, Jingsen (1991). An optimal parallel adaptive sorting algorithm. *Information Processing Letters*, **39**, 195-200.

[CAVA 81a] Cavallo, Roger E. and Klir, George J. (1981). Reconstructability Analysis: Overview and Bibliography. *International Journal of General Systems*, **7**, 1-6.

[CAVA 81b] Cavallo, Roger E. and Klir, George J. (1981). Reconstructability Analysis: Evaluation of Reconstruction Hypothesis. *International Journal of General Systems*, **7**, 7-32.

[CAVA 82] Cavallo, Roger E. and Klir, George J. (1982). Decision Making in Reconstructability Analysis. *International Journal of General Systems*, **8**, 243-255.

[CHA 93] Cha, Shin, Chung, In Sang, and Kwon, Yong Rae (1993). Complexity measures for concurrent programs based on information-theoretic metrics. *Information Processing Letters*, **46**, 43-50.

[CHEN 92] Chen, Calvin C.-Y. and Das, Sajal K. (1992). Breadth-first traversal of trees and integer sorting in parallel. *Information Processing Letters*, **41**, 39-49.

[COHE 91] Cohen, Daniel E. A. (1991). "Introduction to Computer Theory", John Wiley & Sons, Inc., New York.

[COIF 92] Coifman, Ronald R. and Wickerhauser, Mladen Victor (1992). Entropy-Based Algorithms for Best Basis Selection. *IEEE Transactions on Information Theory*, **38**, 713-718.

[CORB 91] Corbett, Petter F. and Scherson, Isaac D. (1991). A unified algorithm for sorting on multidimensional mesh-connected processors. *Information Processing Letters*, **37**, 225-231.

[CUTL 80] Cutland, Nigel (1980). "Computability. An Introduction to Recursive Function Theory", Cambridge University Press, Cambridge.

[DEO 94] Deo, Narsingh, Jain, Amit, and Medidi, Muralidhar (1994). An optimal parallel algorithm for merging using multiselection. *Information Processing Letters*, **50**, 81-87.

[ERIC 88] Erickson, Gary J. and Smith, C. Ray, ed. (1988). "Maximum-Entropy and Bayesian Methods in Science and Engineering." Kluwer Academic Publishers. Dordrecht, The Netherlands.

[FRIE 71] Friedman, Kenneth and Shimony, Abner (1971). Jaynes's Maximum Entropy Prescription and Probability Theory. *Journal of Statistical Physics*, **3**, 381-384.

[GOLU 93] Golub, Gene and Ortega, James M. (1993). "Scientific Computing An Introduction with Parallel Computing", Academic Press, Inc., San Diego, California.

[GOUL 88] Gould, Roland (1988). "Graph Theory", The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California.

[GUIA 85] Guiasu, Silviu, and Shenitzer, Abe (1985). The Principal of Maximum Entropy. *The Mathematical Intelligencer*, **7**, 42-48.

[HAGE 90] Hagerup, Torben and Shen, Hong (1990). Improved Nonconservative Sequential and Parallel Integer Sorting. *Information Processing Letters*, **36**, 57-63.

[HANS 89] Hansen, Augie (1989). "C Programming. A Complete Guide to Mastering the C Language", Addison-Wesley Publishing Company, Inc., Reading Massachusetts.

[HELM 86] Helman, Paul and Veroff, Robert (1986). "Intermediate Problem Solving and Data Structures", The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California.

[HIGA 84] Higashi, Masahiko, Klir, George J. and Pittarelli, Michael A. (1984). Reconstruction Families of Possibilistic Structure Systems. *Fuzzy Sets and Systems*, **12**, 37-60.

[HOEL 71] Hoel, Paul G., Port, Sidney C., and Stone, Charles J. (1971). "Introduction to Statistical Theory", Haughton Mifflin Company, Boston.

[HOPC 79] Hopcroft, John E. and Ullman, Jeffrey, D. (1979). "Introduction to Automata Theory, Languages and Computation", Addison Wesley Publishing Company, Reading, Massachusetts.

[HUNG 90] Hungerford, Thomas (1990). "Abstract Algebra. An Introduction", Saunders College Publishing, Philadelphia.

[HWAN 84] Hwang, Kai and Briggs, Faye' A. (1984). "Computer Architecture and Parallel Processing", McGraw-Hill Publishing Company, New York, New York.

[JONE 82] Jones, Bush (1982). Determination of Reconstruction Families. *International Journal of General Systems*, **8**, 225-228.

[JONE 85a] Jones, Bush (1985). Determination of Unbiased Reconstructions. *International Journal of General Systems*, **10**, 169-176.

[JONE 85b] Jones, Bush (1985). A Greedy Algorithm for a Generalization of the Reconstruction Problem, *International Journal of General Systems*, **11**, 63-68.

[JONE 85c] Jones, Bush (1985). Reconstructability Analysis for General Functions. *International Journal of General Systems*, **11**, 133-142.

[JONE 85d] Jones, Bush (1985). Reconstructability Considerations with Arbitrary Data. *International Journal of General Systems*, **11**, 143-151.

[JONE 85e] Jones, Bush (1985). The Cognitive Content of System Substates. *IEEE Workshop on Languages for Automation.*

[JONE 86] Jones, Bush (1986). *K*-systems Versus Classical Multivariate Systems. *International Journal of General Systems*, **12**, 1-6.

[JONE 89] Jones, Bush (1989). A Program for Reconstructability Analysis. *International Journal of General Systems*, **15**, 199-205.

[KAPU 83] Kapur, J.N. (1983). On Maximum-Entropy Complexity Measures. *International Journal of General Systems*, **9**, 95-102.

[KLIR 86a] Klir, George J. (1986). The Role of Reconstructability Analysis in Social Science Research. *Mathematical Social Sciences*, **12**, 205-225.

[KLIR 86b] Klir, George J. and Pariz Behzad (1986). Relationship Between True and Estimated Possibilistic Systems and their Reconstructions. *International Journal of General Systems*, **12**, 319-331.

[KRIZ 93] Krizanc, Danny (1993). Integer sorting on a mesh-connected array of processors. *Information Processing Letters*, **47**, 283-289.

[KUMA 89] Kumar, Vinod, Kumar, Uma, and Hoshino, Kyuoji (1989). An Application of the Entropy Maximization Approach in Shopping Area Planning. *International Journal of General Systems*, **16**, 25-42.

[LAX 91] Lax, R. F. (1991). "Modern Algebra and Discrete Structrures", Harper Collins Publishers, New York, New York.

[LEIG 92] Leighton, F. Thomson (1992). "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers, San Mateo, California.

[LEVI 78] Levine, R.D. and Tribus, Myron, ed. (1978). "The Maximum Entropy Formalism." The MIT Press, Cambridge, Massachusetts.

[LEWI 59] Lewis II, P. M. (1959). Approximating Probability Distributions to Reduce Storage Requirements. *Information and Control*, **2**, 214-225.

[LEWI 81] Lewis, Harry R. and Papadimitriou, Christos H. (1981). "Elements of the Theory of Computation", Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[LIN 90] Lin, Chau-Jy (1990). Parallel Algorithm for Generating Permutations on Linear Array. *Information Processing Letters*. **35**, 167-170.

[LIU 92] Liu, Xiaoqing and Kim, Junguk L. (1992). An efficient parallel sorting algorithm. *Information Processing Letters*, **43**, 129-133.

[LUCH 93] Luchetti, Carlo and Pinotti, M. Cristina (1993). Some somments on building heaps in parallel. *Information Processing Letters*, **47**, 145-148.

[MIKL 84] Miklosko, J. and Kotov, V. E. , ed. (1984). "Algorithms, Software and Hardware of Parallel Computers. VEDA, Publishing House of the Slovak Academy of Sciences, Bratislava, Czechoslovakia.

[MILL 88a] Miller, R., Prasanna-Kumar, V.K., Reisis, D., and Stout, Q.F. (1988). Data Movement Operations and Applications on Reconfigurable VLSI Arrays. *Proceedings of the Fifth MIT Conference on Advanced Research in VLSI*, 163-178.

[MILL 88b] Miller, R., Prasanna-Kumar, V.K., Reisis, D., and Stout, Q.F. (1988). Meshes with Reconfigurable Buses. *Proceedings of the Internatinal Conference on Parallel Processing*, **1**, 205-208.

[MOLL 86] Molluzzo, John C. and Buckley, Fred (1986). "A First Course in Discrete Mathematics", Wadsworth Publishing Company, Belmont, California.

[NAPS 92] Naps, Thomas L. and Pothering, George J. (1992). "Introduction to Data Structures and Algorithms Analysis with Pascal", West Publishing Company, St. Paul.

[OLAR 91] Olariu, S., Schwing, J.L., and Zhang, J. (1991). Fundamental Algorithms on Reconfigurable Meshes. *Proceedings of the 29th Allerton Conference on Communications, Control, and Computing*, 811 - 820.

[PINO 91] Pinotti, Maria Cristina and Pucci, Geppino (1991). Parallel Priority Queues. *Information Processing Letters*, **40**, 33-40.

[PITT 89] Pittarelli, Michael (1989). Uncertainty and Estimation in Reconstructability Analysis. *International Journal of General Systems*, **15**, 1-58.

[PITT 90] Pittarelli, Michael (1990). Reconstructability Analysis Using Probability Intervals. *International Journal of General Systems*, **16**, 215-233.

[POLI 93] Politis, Dimitris Nicolas (1993). Nonparametric Maximum Entropy. *IEEE Transactions on Information Theory*, **39**, 1409-1413.

[PREP 85] Preparata, Franco P. and Shamos, Michael Ian (1985). "Computatinal Geometry. An Introduction", Springer-Verlag, New York.

[QUIN 87] Quinn, Michael J. (1987). "Designing Efficient Algorithms for Parallel Computers", McGraw-Hill Book Company, New York, New York.

[RAME 86] Ramer, Arthur (1986). Transversals and Beta-Property in the Reconstructability Theory. *International Journal of General Systems*, **13**, 39-45.

[RAO 91] Rao, Nageswara S.V. and Zhang, Weixiong (1991). Building heaps in parallel. *Information Processing Letters*, **37**, 355-358.

[ROGE 87] Rogers, Hartley Jr. (1987). "Theory of Recursive Functions and Efficient Computability", The MIT Press, Cambridge, Massachusetts.

[ROSS 88] Ross, Sheldon (1988). "A First Course in Probability", Macmillan Publishing Company, New York.

[SEID 86] Seidenfeld, Teddy (1986). Entropy and Uncertainty. *Philosophy of Science*, **53**, 467-491.

[SHOR 80] Shore, John E. and Johnson, Rodney W. (1980). Axiomatic Derivation of the Principle of Maximum Entropy and the Principal of Minimum Cross-Entropy. *IEEE Transactions on Information Theory*, **IT-26**, 26-37.

[SMIT 90] Smith, Douglas, Eggen, Maurice, and St. Andre, Richard (1990). "A Transition to Advanced Mathematics", Brooks/Cole Publishing Company, Pacific Grove, California.

[STAM 93] Stamoulis, George D. and Tsitsiklis, John N. (1993). An efficient algorithm for multiple simultaneous broadcasts in the hypercube. *Information Processing Letters*, **46**, 219-224.

[STAN 86] Stanley, Richard P. (1986). "Enumerative Combinatorics Volume I", Wadsworth & Brooks/Cole, Monterey, California.

[STEV 90] Stevens, W. Richard (1990). "Unix Network Programming", Prentice Hall, Englewood Cliffs, New Jersey.

[STOU 86] Stout, Q.F. (1986). Meshes with Multiple Buses. *Proceedings of the 27th Symposium on the Foundations of Computer Science*, 264 - 273.

[TAYL 89] Taylor, Stephen (1989). "Parallel Logic Programming Techniques", Prentice Hall, Englewood Cliffs, New Jersey.

[TCHU 91] Tchuente, Maurice (1991). "Parallel Computation on Regular Arrays", Manchester University Press, Manchester, England.

[TRIV 93] Trivedi, Sudhir K. *Reconstructability Theory for General Systems and its Application to Automated Rule Learning.* Ph.D. dissertation, Louisiana State University, Baton Rouge, LA, 1993.

[TRIV 94] Trivedi, Sudhir K., Jones, J. Bush, and Iyengar, Sithirama (1994). Reconstruction of Possibilistic Systems with Incomplete Information. To be published in the *Proceedings of the 32nd Southeast ACM Conference.*

[VALD 83] Valdes-Perez, Raul E. and Conant Roger C. (1983). Information Loss Due to Data Quantization in Reconstructability Analysis. *International Journal of General Systems*, **9**, 235-247.

[WEN 91] Wen, Zhaofang (1991). Parallel multiple search. *Information Processing Letters*, **37**, 181-186.

Appendix A

Terminology

Adaptive Algorithm

An adaptive algorithm if it can efficiently solve a problem of any input size, assuming sufficient memory space is available.

Attributes

Set of values that variables of a system my take on.

Behavior System

"the sixtuple:

$$"B = (V, \mathscr{V}, s, A, Q, f) \quad (1)$$

where $V = \{v_i | \, i \in N_n\}$ ($N_n = \{1,2,...,n\}$) is a set of variables, $\mathscr{V} = \{V_j | \, j \in N_m, m \leq n\}$ is a family of state sets; $V \rightarrow \mathscr{V}$ is an onto assignment function by which one state set from $\mathscr{V}$ is assigned to each variable in V; $A = s(v_1) \times s(v_2) \times ... \times s(v_n)$ is the set of all potential aggregate states; Q is a set of real numbers which includes 0; f: $A \rightarrow Q$ is a function, referred to as a behavior function, which represents information regarding the aggregate states of the behavior system" [CAVA 81b].

Cognitive Content

"The contribution of knowledge about the substate to the overall system behavior" [JONE 85e].

Coarse-Grain processors

Coarse-grain processors are ones with "fairly sophisticated computational power and potentially large memories" [LEIG 92].

Entropy

Entropy is the amount of uncertainty contained in a probability distribution: $H_m(\bar{p}) = H_m(p_1, p_2, ... p_m) = -\sum_{k=1}^{m} p_k \ln p_k$.

Equivalence Classes

The set of G-structures (any reconstruction hypothesis which is internally consistent, irredundant and contains all relevant variables) may be partitioned into equivalence classes of non-overlapping sets of G-structures [CAVA 81a].

Godel Numbering

Godel numbering is an enumeration technique that maps strings to the set of non-negative integers.

G-Systems (General Systems)

Structure systems with general functions which need not be behavior functions. The set of behavior systems is a subset of structure systems.

Identification problem

"the identification of specified properties of an overall system from appropriate properties of given subsystems" [CAVA 81a].

K-Systems (Klir Systems)

A behavior system that is induced from a *g*-system via a type of isomorphism [JONE 85c].

Maximum Entropy

Let $f_s : A \rightarrow Q$ be a behavior function which essentially chooses one system from the reconstruction family. The set of values $\{f_s(\alpha) | \alpha \in A$ has the maximum entropy from among all such sets associated with overall systems in the reconstruction family if it meets the requirement that

$$[f_s \downarrow^k V] =^k f =^k [f \downarrow^k V] \quad .$$

Multiple Data Architecture

An architectural design in which each processing element (PE) has its own local memory (e.g. SIMD and MIMD architectures).

Null Extensions

"Let β be a substate. Then an overall state α is the null extension of β, written β', if $\alpha \rangle \beta$ and every variable of α which does not occur in β has a value of zero" [JONE 85b].

Potential Aggregate State Orderings

The potential aggregate state orderings is a one-to-one correspondence between the set of all possible states in the overall system and strings that contain the ordering of the corresponding variable values

Projection

"Let $[f \downarrow Z\ ]$ denote the projection of f which disregards all variables in V except those in set $Z \subset V$. Then, $[f \downarrow Z\ ]$ is itself a mapping from a set of states (substates of states in A) to Q:

$$[f \downarrow Z] : \underset{v_j \in Z}{\times} s(v_i) \rightarrow Q$$

such that

$$[f \downarrow Z](\beta) = g(\{f(\alpha)|\alpha\rangle\beta\}),$$

where function g is determined by the nature of function f" [CAVA 81b].

Reconfigurable Bus System

A reconfigurable bus system in one in which the configuration can be dynamically altered.

Reconstructability Analysis

"the process of investigating the possibilities of reconstructing desirable properties of overall systems from the knowledge of the corresponding properties of their various subsystems " [CAVA 81b] (i.e. the solution processes associated with the identification problem and the reconstruction problem [CAVA 81a]).

Reconstruction Family

"Given a particular reconstruction hypothesis, say S, there exists a family of overall systems which are compatible with the hypothesis in the sense that relevant projections of the behavior functions of each of these overall systems are exactly those behavior functions which are included in the reconstruction hypothesis". "... we denote this family as $\mathfrak{F}_S$" [CAVA 81b].

Reconstruction Hypothesis

"meaningful sets of subsystems or structure models" [CAVA 81a].

Reconstruction Problem

"the problem of determining which subsystems of an overall system are adequate in the sense that specific properties of the overall system can be reconstructed, with a desirable level of approximation, from the knowledge of the corresponding properties of the involved subsystems" [CAVA 81a].

Structure Systems

"the set of all reconstruction hypotheses" [CAVA 81a].

Substates

"For each aggregate state (n-tuple)

$$\alpha = (\alpha_i | i \in N_n) \in A$$

of a behavior system defined by (1) and for each state

$$\beta = (\beta_j | j \in X, X \subset N_n)$$

associated with variables in set

$$Z = \{v_j | j \in X, X \subset N_n\} \subset V,$$

let β be called a substate of α (or α be called a superstate of β) if and only if

$$\beta_j = \alpha_j \text{ for all } j \in X.$$

Let $\beta\langle\alpha$ and $(\alpha\rangle\beta)$ denote that β is a substate of α" [CAVA 81b].

Subsystem

"given an overall system, each non-empty subset of the set of variables identifies one subsystem of that overall system" [CAVA 81b].

Systems Modelling

"a complex conglomerate of interrelated systems problems. It can loosely be characterized as the process of determining a system on an object of investigation which is an adequate model of the relevant phenomena associated with the object" [CAVA 81a].

Unbiased Reconstruction

The behavior function chosen as the maximum entropy probability distribution is referred to as the unbiased reconstruction from the reconstruction family [CAVA 81b].

Very-Fine-Grain Processors

Very-fine-grain processors are ones that have at most a few registers of memory [LEIG 92].

Appendix B

Index of Notation

p	number of processors
l	number of variables in the overall system
Σ	set of possible variable values; $\Sigma = \cup_i \bar{s}(v_i)$
v_i	variable
$s(v_i)$	s: $v_i \rightarrow$ set of possible variable values for the variable v_i
f	the behavior function of the overall system
α	state of the overall system
β	substate of the overall system
$\langle$	substate
$\rangle$	superstate
$\downarrow$	projection
β'	null extension of substate β
^{m}f	the behavior function of a substate
$\hat{f}$	approximation to f
a	$a_i = \Sigma_j f_{ij}$
$\hat{a}$	approximation to a; $\hat{a}_i = \Sigma_j \hat{f}_{ij}$
P	the set of potential aggregate state orderings
D	the set of substates in the structure system
n	number of states in the overall system

gn(w) godel number for the string w

$\bar{s}(v_i)$ set of orderings for the values that variables can have plus the value of zero

E set of independent substates

C number of optimal partitions of states into disjoint sets

Appendix C

Evaluation of Structural Complexities

C.1 Evaluation of the gn Routine

lines	structure type	structural complexity
5		O(1)
6		O(1)
7		O(*l*)
8		O(1)
9		O(1)
7 - 10	iteration	O(*l*)
11		O(1)
4 - 12	sequence	O(*l*)
1 - 12	routine	O(*l*)

C.2 Evaluation of the gn^{-1} Routine

lines	structure type	structural complexity
4		O(*l*)
5		O(1)
6		O(1)
4 - 7	iteration	O(*l*)
8		O(1)
3 - 9	sequence	O(*l*)

lines	structure type	structural complexity
1 - 9	routine	$O(l)$

C.3 Evaluation of the Algorithm Initialization

lines	structure type	structural complexity
3		$O(\frac{\|P\|}{p})$
4		$O(l)$
5		$O(1)$
6		$O(l)$
7		$O(1)$
8		$O(1)$
7 - 9	selection	$O(1)$
6 - 10	iteration	$O(l)$
3 - 11	iteration	$O(\frac{l\|P\|}{p})$
1 - 12	parallel	$O(\frac{l\|P\|}{p})$
13		$O(\|\mathbf{D}\|)$
14		$O(l)$
15		$O(1)$
16		$O(1)$
17		$O(1)$
19		$O(\|\Sigma\|)$
16 - 20	selection	$O(\|\Sigma\|)$
14 - 21	iteration	$O(l\|\Sigma\|)$

lines		structure type	structural complexity
	22		$O(l)$
	23		$O(l)$
13 -	24	iteration	$O(\mathbf{D}l\lvert\Sigma\rvert)$
1 -	24	algorithm	$O(\frac{\lvert P\rvert}{p}l+\lvert D\rvert l\lvert\Sigma\rvert)$

C.4 Evaluation of Generating Independent Substates in Parallel

lines		structure type	structural complexity
	1		$O(1)$
	4		$O(\frac{\lvert P\rvert}{p})$
	5		$O(1)$
	6		$O(1)$
	7		$O(l)$
	8		$O(l)$
	9		$O(1)$
	10		$O(1)$
9 -	11	selection	$O(1)$
8 -	12	iteration	$O(l)$
	13		$O(\frac{\lvert D\rvert}{\lvert P\rvert}p)$
	14		$O(n)$
	15		$O(n)$
	16		$O(1)$
14 -	17	selection	$O(n)$

lines	structure type	structural complexity
13 - 18	critical section	$O(\frac{\|D\|}{\|P\|}p)$
6 - 19	selection	$O(\frac{\|D\|}{\|P\|}p)$
4 - 20	iteration	$O(\|\mathbf{D}\|n)$
2 - 21	parallel	$O(\|\mathbf{D}\|n)$
1 - 21	algorithm	$O(\|\mathbf{D}\|n)$

C.5 Evaluation of Partitioning the States into Disjoint Sets in Parallel

lines	structure type	structural complexity
1		O(E)
2		O(1)
3		O(1)
1 - 4	iteration	O(E)
5		O(1)
8		$O(\frac{\|P\|}{p})$
9		O(1)
10		O(1)
11		$O(l + \|\Sigma\|^l)$
12		O(1)
13		O(1)
14		$O(\frac{E}{\|P\|}p)$
15		O(E)

lines		structure type	structural complexity
	16		$O(n)$
	17		$O(1)$
	19		$O(1)$
16 -	20	selection	$O(n)$
15 -	21	iteration	$O(En)$
	22		$O(n)$
13 -	23	critical section	$O(E^2n)$
	24		$O(1)$
	25		$O(1)$
24 -	26	selection	$O(1)$
	27		$O(1)$
10 -	28	selection	$O(E^2n)$
8 -	29	iteration	$O(E^2n)$
6 -	30	parallel	$O(E^2n)$
	31		$O(l)$
	32		$O(1)$
31 -	33	iteration	$O(l)$
	34		$O(\lvert\Sigma\rvert^l)$
	35		$O(E)$
	36		$O(n)$

lines	structure type	structural complexity
35 - 37	iteration	O(En)
38		O(E)
1 - 38	algorithm	O(E^2n)

C.6 Evaluation of the GenerateSuperstates Routine

The GenerateSuperstates routine is taken to have the structural complexity of O($|\Sigma|^l$). Justification for this order of magnitude is as follows. If the word passed to the routine has no zero elements, then the routine generates a set of size 0 ($|\Sigma|^0 = 1$). If the word contains exactly one zero element, then the routine generates a set of size $|\Sigma|$ ($|\Sigma|^1 = |\Sigma|$). If the word contains exactly two zero elements, then the routine generates a set of size $|\Sigma|^2$, etc. In the worst case, the word contains all zero elements, in which the routine generates a set of size $|\Sigma|^l$. Therefore, the structural complexity of the GenerateSuperstates routine is taken to be O($|\Sigma|^l$).

C.7 Evaluation of the Parallelization of the Reconstruction Process

lines	structure type	structural complexity
3		O($\frac{\|P\|}{p}$)
4		O(1)
5		O(1)
4 - 6	selection	O(1)
3 - 7	iteration	O($\frac{\|P\|}{p}$)
1 - 8	parallel	O($\frac{\|P\|}{p}$)

lines		structure type	structural complexity
	9		$O(1)$
	10		$O(\delta_U)$
	11		$O(C)$
	14		$O(\frac{\|P\|}{p})$
	15		$O(1)$
	16		$O(1)$
	17		$O(1)$
	18		$O(n)$
	19		$O(1)$
	20		$O(1)$
	21		$O(1)$
	22		$O(1)$
18 -	23	iteration	$O(n)$
15 -	24	selection	$O(n)$
14 -	25	iteration	$O(\frac{\|P\|}{p}n)$
12 -	26	parallel	$O(\frac{\|P\|}{p}n)$
	27		$O(1)$
	28		$O(1)$
	29		$O(n)$
	30		$O(1)$
	31		$O(1)$

lines		structure type	structural complexity
	32		$O(l)$
	33		$O(l)$
29 -	34	iteration	$O(nl)$
	37		$O(\frac{\|P\|}{p})$
	38		$O(1)$
	39		$O(1)$
	40		$O(1)$
	42		$O(1)$
39 -	43	selection	$O(1)$
38 -	44	selection	$O(1)$
37 -	45	iteration	$O(\frac{\|P\|}{p})$
35 -	46	parallel	$O(\frac{\|P\|}{p})$
11 -	47	iteration	$O(C\frac{\|P\|}{p}n)$
	48		$O(1)$
	51		$O(\frac{\|P\|}{p})$
	52		$O(1)$
	53		$O(1)$
	54		$O(1)$
52 -	55	selection	$O(1)$
51 -	56	iteration	$O(\frac{\|P\|}{p})$

lines	structure type	structural complexity
49 - 57	parallel	$O(\frac{\|P\|}{p})$
10 - 58	iteration	$O(\delta_U C \frac{\|P\|}{p} n)$
1 - 58	algorithm	$O(\delta_U C \frac{\|P\|}{p} n)$

C.8 Evaluation of the Parallelization of a Greedy Algorithm for a Generalization of the Unbiased Reconstruction

lines	structure type	structural complexity
1		$O(1)$
4		$O(\frac{\|P\|}{p})$
5		$O(1)$
6		$O(l)$
5 - 7	selection	$O(l)$
4 - 8	iteration	$O(\frac{\|P\|}{p} l)$
2 - 9	parallel	$O(\frac{\|P\|}{p} l)$
12		$O(\frac{\|P\|}{p})$
13		$O(1)$
14		$O(1)$
13 - 15	selection	$O(1)$
12 - 16	iteration	$O(\frac{\|P\|}{p})$
10 - 17	parallel	$O(\frac{\|P\|}{p})$
18		$O(1)$
19		$O(1)$

lines		structure type	structural complexity
	20		O(1)
	21		O(δ_G)
	22		O(1)
	23		O(|**D**|)
	24		O(1)
	25		O(p)
	26		O(1)
25 -	27	iteration	O(p)
	28		O(1)
	31		O(l)
	32		O(l)
	33		O(1)
29 -	34	parallel	O(l)
	37		O(1)
	38		O(1)
	39		O(1)
	41		O(1)
	42		O(1)
37 -	43	selection	O(1)
	44		O(1)
	45		O(1)

lines		structure type	structural complexity
	46		O(1)
35 -	47	parallel	O(1)
	50		O(1)
	51		O (1)
48 -	52	parallel	O(1)
	55		O(1)
	56		O(1)
	57		O(1)
	59		O(1)
	60		O(1)
	61		O(1)
59 -	62	selection	O(1)
55 -	63	selection	O(1)
	64		O(1)
	65		O(1)
53 -	66	parallel	O(1)
	68		O(1)
23 -	69	iteration	$O(\|\mathbf{D}\|p + \|\mathbf{D}\|l)$
	70		$O(\|\mathbf{D}\|)$
	71		$O(\|\mathbf{D}\|)$
	72		$O(\delta_U C\frac{\|P\|}{p}n)$

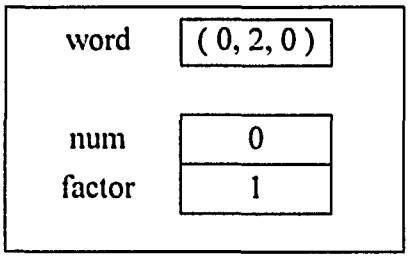
lines		structure type	structural complexity
	73		$O(1)$
	76		$O(\frac{\|P\|}{p})$
	77		$O(1)$
76 -	78	iteration	$O(\frac{\|P\|}{p})$
74 -	79	parallel	$O(\frac{\|P\|}{p})$
21 -	80	iteration	$O(\delta_G \delta_U C \frac{\|P\|}{p} n)$
1 -	80	algorithm	$O(\delta_G \delta_U C \frac{\|P\|}{p} n)$

Appendix D

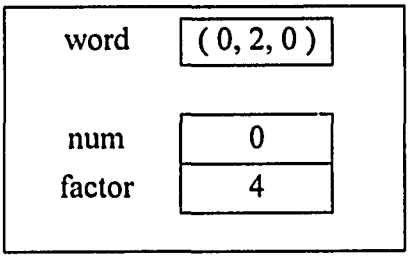
Algorithm Iteration

D.1 Iteration of the gn Routine

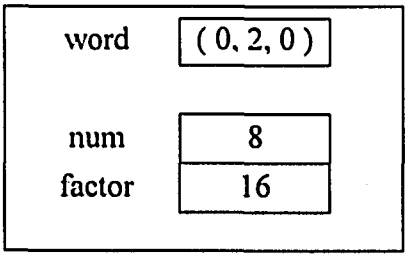
To illustrate the iteration of the gn routine, two words in the example structure system are selected. First, we trace the iteration of the gn routine when the word passed in as a parameter is the substate represented by the ordering (0, 2, 0). Note that in the example structure system $\Sigma = \{ 0, 1, 2, 3 \}$. Therefore, $|\Sigma| = 4$.



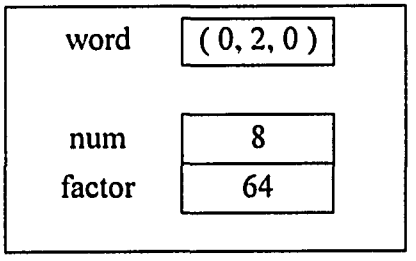
initially:



i = 3



i = 2

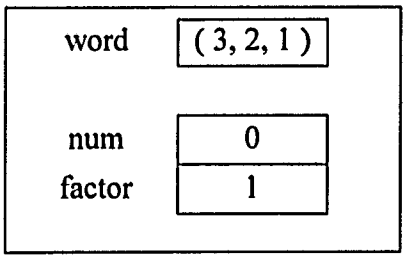


i = 1

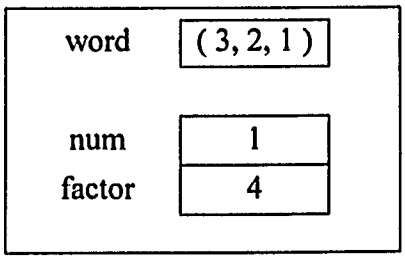
The functional value of 8 is returned at the completion of iteration.

Second, we trace the iteration of the gn routine when the word passed in as a parameter is the state represented by the ordering (3, 2, 1).

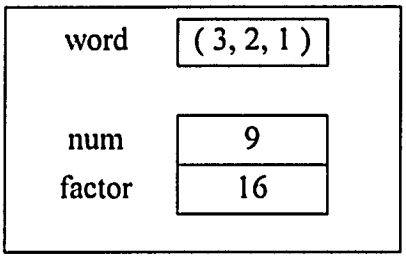
initially:



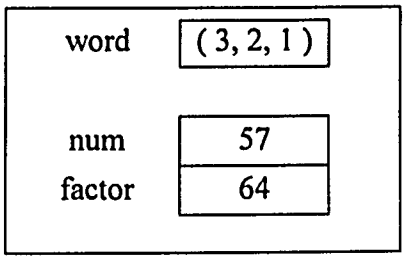
i = 3



i = 2



i = 1

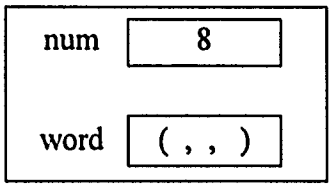


The functional value of 57 is returned at the completion of iteration.

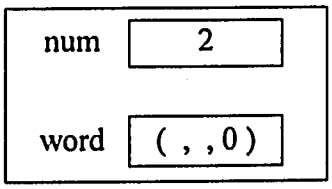
D.2 Iteration of the gn^{-1} Routine

To illustrate the iteration of the gn^{-1} routine, two numbers in the range of the example structure system are selected. First, we trace the iteration of the gn^{-1} routine when the number passed in as a parameter is the number 8. Note that in the example structure system $\Sigma = \{ 0, 1, 2, 3 \}$. Therefore, $|\Sigma| = 4$.

initially:



i = 3



i = 2

num	0
word	(, 2, 0)

i = 1

num	0
word	(0, 2, 0)

The functional value of (0, 2, 0) is returned at the completion of iteration.

Second, we trace the iteration of the gn^{-1} routine when the number passed in as a parameter is the number 57.

initially:

num	57
word	(, ,)

i = 3

num	14
word	(, , 1)

i = 2

num	3
word	(, 2, 1)

i = 1

num	0
word	(3, 2, 1)

The functional value of (3, 2, 1) is returned at the completion of iteration.

D.3 Iteration of the Algorithm Initialization

Upon the iteration of the algorithm initialization, the local memory has initialized the array state. Each processor initializes all $\left\lceil \frac{|P|}{p} \right\rceil$ elements of the array. All array elements corresponding to states in the overall system are initialized to 1. All other array elements are initialized to -1. This initialization is performed in

parallel. Once this initialization is complete, the input, which is necessarily sequential, of behavior function values is performed. The behavior function values are stored in the *f* array in local memory at the corresponding substate position. The state value at that position is changed to a value of 0. Upon completion of the input, only those elements of the *f* array that correspond to substates in the structure system have values, and only those elements contain a state value of 0. All state element values that contain a value of -1 are either substates of the overall system that are not present in our structure system, or they are not substates of the overall system due to the gn function not being onto.

Upon completion of this iteraiton, the local memory for the example structure system is initialized as follows:

PE_0	state	*f*
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	0	0.17
6	0	0.16
7	0	0.12

PE_1	state	*f*
0	-1	
1	0	0.14
2	0	0.18
3	0	0.23
4	-1	
5	-1	
6	-1	
7	-1	

PE_2	state	*f*
0	-1	
1	0	0.11
2	0	0.14
3	0	0.18
4	0	0.25
5	1	
6	1	
7	1	

PE_3	state	f
0	0	0.18
1	1	
2	1	
3	1	
4	-1	
5	1	
6	1	
7	1	

PE_4	state	f
0	-1	
1	0	0.2
2	0	0.2
3	0	0.17
4	0	0.2
5	1	
6	1	
7	1	

PE_5	state	f
0	0	0.37
1	1	
2	1	
3	1	
4	-1	
5	-1	
6	-1	
7	-1	

PE_6	state	f
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

PE_7	state	f
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

D.4 Iteration of Generating Independent Substates in Parallel

Once the local array state has been initialized, we are ready to progress to generating independent substates in parallel. The main memory set variables E and NullExtension are initialized to the empty set. At each iteration, if the state value indexed in conjunction with the iteration number has a value of 0, then the gn^{-1} of the corresponding godel number is found to generate a word which represents a substate in the structure system. The null extension of this word is determined and if the null

extension is not found in the main memory set NullExtension, then it is added to the main memory set *E*, and its null extension is added to the main memory set NullExtension. When the iteration is completed, the main memory set *E* contains a set of independent substates. As stated in section 1.2.1.2.3, this set is not unique.

initially:

PE_0	state	*E*
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	0	
6	0	
7	0	

word

PE_1	state	*E*
0	-1	
1	0	
2	0	
3	0	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_2	state	*E*
0	-1	
1	0	
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word

PE_3	state	*E*
0	0	
1	1	
2	1	
3	1	
4	-1	
5	1	
6	1	
7	1	

word

PE_4	state	*E*
0	-1	
1	0	
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word

PE_5	state	*E*
0	0	
1	1	
2	1	
3	1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_6	state	E
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	-1	
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

NullExtension { (1,1,1) }

j = 0

PE_0	state	E
0	**-1**	**0**
1	-1	
2	-1	
3	-1	
4	-1	
5	0	
6	0	
7	0	

word

PE_1	state	E
0	**-1**	**0**
1	0	
2	0	
3	0	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_2	state	E
0	**-1**	**0**
1	0	
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word

PE_3	state	E
0	**0**	**1**
1	1	
2	1	
3	1	
4	-1	
5	1	
6	1	
7	1	

word **(1,2,1)**

PE_4	state	E
0	**-1**	**0**
1	0	
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word

PE_5	state	E
0	**0**	**1**
1	1	
2	1	
3	1	
4	-1	
5	-1	
6	-1	
7	-1	

word **(2,2,1)**

PE_6	state	E
0	**-1**	**0**
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	**-1**	**0**
1	-1	
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

NullExtension **{ (1,1,1), (1,2,1), (2,2,1) }**

j = 1

PE_0	state	*E*
0	-1	0
1	**-1**	**0**
2	-1	
3	-1	
4	-1	
5	0	
6	0	
7	0	

word

PE_1	state	*E*
0	-1	0
1	**0**	**0**
2	0	
3	0	
4	-1	
5	-1	
6	-1	
7	-1	

word **(1,2,1)**

PE_2	state	*E*
0	-1	0
1	**0**	**0**
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word **(1,1,1)**

PE_3	state	*E*
0	0	1
1	**1**	**0**
2	1	
3	1	
4	-1	
5	1	
6	1	
7	1	

word

PE_4	state	*E*
0	-1	0
1	**0**	**1**
2	0	
3	0	
4	0	
5	1	
6	1	
7	1	

word **(2,1,1)**

PE_5	state	*E*
0	0	1
1	**1**	**0**
2	1	
3	1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_6	state	E
0	-1	0
1	**-1**	**0**
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	-1	0
1	**-1**	**0**
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

NullExtension { (1,1,1), (1,2,1), (2,2,1), **(2,1,1)** }

j = 2

PE_0	state	E
0	-1	0
1	-1	0
2	**-1**	**0**
3	-1	
4	-1	
5	0	
6	0	
7	0	

word

PE_1	state	E
0	-1	0
1	0	0
2	**0**	**1**
3	0	
4	-1	
5	-1	
6	-1	
7	-1	

word **(1,2,2)**

PE_2	state	E
0	-1	0
1	0	0
2	**0**	**1**
3	0	
4	0	
5	1	
6	1	
7	1	

word **(1,1,2)**

PE_3	state	E
0	0	1
1	1	0
2	**1**	**0**
3	1	
4	-1	
5	1	
6	1	
7	1	

word

PE_4	state	E
0	-1	0
1	0	1
2	**0**	**1**
3	0	
4	0	
5	1	
6	1	
7	1	

word **(2,1,2)**

PE_5	state	E
0	0	1
1	1	0
2	**1**	**0**
3	1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_6	state	E
0	-1	0
1	-1	0
2	**-1**	**0**
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	-1	0
1	-1	0
2	**-1**	**0**
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	

word

NullExtension { (1,1,1), (1,2,1), (2,2,1), (2,1,1), **(1,2,2), (1,1,2), (2,1,2)** }

j = 3

PE_0	state	*E*
0	-1	0
1	-1	0
2	-1	0
3	**-1**	**0**
4	-1	
5	0	
6	0	
7	0	

word

PE_1	state	*E*
0	-1	0
1	0	0
2	0	1
3	**0**	**1**
4	-1	
5	-1	
6	-1	
7	-1	

word **(1,2,3)**

PE_2	state	*E*
0	-1	0
1	0	0
2	0	1
3	**0**	**1**
4	0	
5	1	
6	1	
7	1	

word **(1,1,3)**

PE_3	state	*E*
0	0	1
1	1	0
2	1	0
3	**1**	**0**
4	-1	
5	1	
6	1	
7	1	

word

PE_4	state	*E*
0	-1	0
1	0	1
2	0	1
3	**0**	**1**
4	0	
5	1	
6	1	
7	1	

word **(2,1,3)**

PE_5	state	*E*
0	0	1
1	1	0
2	1	0
3	**1**	**0**
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_6	state	E
0	-1	0
1	-1	0
2	-1	0
3	**-1**	**0**
4	-1	
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	-1	0
1	-1	0
2	-1	0
3	**-1**	**0**
4	-1	
5	-1	
6	-1	
7	-1	

word

NullExtension { (1,1,1), (1,2,1), (2,2,1), (2,1,1), (1,2,2), (1,1,2), (2,1,2), **(1,2,3), (1,1,3), (2,1,3)**}

j = 4

PE_0	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	**-1**	**0**
5	0	
6	0	
7	0	

word

PE_1	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	**-1**	**0**
5	-1	
6	-1	
7	-1	

word

PE_2	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	**0**	**0**
5	1	
6	1	
7	1	

word **(1,1,1)**

PE_3	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	**-1**	**0**
5	1	
6	1	
7	1	

word

PE_4	state	E
0	-1	0
1	0	1
2	0	1
3	0	1
4	**0**	**0**
5	1	
6	1	
7	1	

word **(2,1,1)**

PE_5	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	**-1**	**0**
5	-1	
6	-1	
7	-1	

word

PE_6	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	**-1**	**0**
5	-1	
6	-1	
7	-1	

word

PE_7	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	**-1**	**0**
5	-1	
6	-1	
7	-1	

word

NullExtension { (1,1,1), (1,2,1), (2,2,1), (2,1,1), (1,2,2), (1,1,2), (2,1,2), (1,2,3), (1,1,3), (2,1,3)}

j = 5

PE_0	state	*E*
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	**0**	**-1**
6	0	
7	0	

word **(1,1,1)**

PE_1	state	*E*
0	-1	0
1	0	0
2	0	1
3	0	1
4	-1	0
5	**-1**	**0**
6	-1	
7	-1	

word

PE_2	state	*E*
0	-1	0
1	0	0
2	0	1
3	0	1
4	0	0
5	**1**	**0**
6	1	
7	1	

word

PE_3	state	*E*
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	**1**	**0**
6	1	
7	1	

word

PE_4	state	*E*
0	-1	0
1	0	1
2	0	1
3	0	1
4	0	0
5	**1**	**0**
6	1	
7	1	

word

PE_5	state	*E*
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	**-1**	**0**
6	-1	
7	-1	

word

PE_6	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	**-1**	**0**
6	-1	
7	-1	

word

PE_7	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	**-1**	**0**
6	-1	
7	-1	

word

NullExtension { (1,1,1), (1,2,1), (2,2,1), (2,1,1), (1,2,2), (1,1,2), (2,1,2), (1,2,3), (1,1,3), (2,1,3) }

j = 6

PE_0	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	0	-1
6	**0**	**0**
7	0	

word **(1,1,2)**

PE_1	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	-1	0
5	-1	0
6	**-1**	**0**
7	-1	

word

PE_2	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	0	0
5	1	0
6	**1**	**0**
7	1	

word

PE_3	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	1	0
6	**1**	**0**
7	1	

word

PE_4	state	E
0	-1	0
1	0	1
2	0	1
3	0	1
4	0	0
5	1	0
6	**1**	**0**
7	1	

word

PE_5	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	-1	0
6	**-1**	**0**
7	-1	

word

PE_6	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	**-1**	**0**
7	-1	

word

PE_7	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	**-1**	**0**
7	-1	

word

NullExtension	{ (1,1,1), (1,2,1), (2,2,1), (2,1,1), (1,2,2), (1,1,2), (2,1,2), (1,2,3), (1,1,3), (2,1,3) }

j = 7

PE_0	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	0	-1
6	0	0
7	**0**	**0**

word **(1,1,3)**

PE_1	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	-1	0
5	-1	0
6	-1	0
7	**-1**	**0**

word

PE_2	state	E
0	-1	0
1	0	0
2	0	1
3	0	1
4	0	0
5	1	0
6	1	0
7	**1**	**0**

word

PE_3	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	1	0
6	1	0
7	**1**	**0**

word

PE_4	state	E
0	-1	0
1	0	1
2	0	1
3	0	1
4	0	0
5	1	0
6	1	0
7	**1**	**0**

word

PE_5	state	E
0	0	1
1	1	0
2	1	0
3	1	0
4	-1	0
5	-1	0
6	-1	0
7	**-1**	**0**

word

PE_6	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	-1	0
7	-1	0

word

PE_7	state	E
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
5	-1	0
6	-1	0
7	-1	0

word

NullExtension	{ (1,1,1), (1,2,1), (2,2,1), (2,1,1), (1,2,2), (1,1,2), (2,1,2), (1,2,3), (1,1,3), (2,1,3) }

D.5 Iteration of Partitioning the States into Disjoint Sets in Parallel

To illustrate the iteration of partitioning the states into disjoint sets in parallel, we use the set of independent states computed in section D.4.

PE_0	state	f	E	C	Superstates
0	-1		0	0	
1	-1		0	0	
2	-1		0	0	
3	-1		0	0	
4	-1		0	0	
5	0	0.17	0	0	
6	0	0.16	0	0	
7	0	0.12	0	0	

PE_1	state	f	E	C	Superstates
0	-1		0	0	
1	0	0.14	0	0	
2	0	0.18	1	2	{(1,2,2), (2,2,2)}
3	0	0.23	1	2	{(1,2,3), (2,2,3)}
4	-1		0	0	
5	-1		0	0	
6	-1		0	0	
7	-1		0	0	

PE_2	state	f	E	C	Superstates
0	-1		0	0	
1	0	0.11	0	0	
2	0	0.14	1	3	{(1,1,2), (1,2,2)}
3	0	0.18	1	3	{(1,1,3), (1,2,3)}
4	0	0.25	0	0	
5	-1		0	0	
6	-1		0	0	
7	-1		0	0	

PE_3	state	f	E	C	Superstates
0	0	0.18	1	1	{(1,2,1), (1,2,2), (1,2,3)}
1	1		0		
2	1		0		
3	1		0		
4	-1		0		
5	1		0		
6	1		0		
7	1		0		

PE_4	state	f	E	C	Superstates
0	-1		0		
1	0	0.2	1	2	{(2,1,1), (2,2,1)}
2	0	0.2	1	3	{(2,1,2), (2,2,2)}
3	0	0.17	1	3	{(2,1,3), (2,2,3)}
4	0	0.2	0		
5	1		0		
6	1		0		
7	1		0		

PE_5	state	f	E	C	Superstates
0	0	0.37	1	1	{(2,2,1), (2,2,2), (2,2,3)}
1	1		0	0	
2	1		0	0	
3	1		0	0	
4	-1		0	0	
5	-1		0	0	
6	-1		0	0	
7	-1		0	0	

PE_6	state	f	E	C	Superstates
0	-1		0	0	
1	-1		0	0	
2	-1		0	0	
3	-1		0	0	
4	-1		0	0	
5	-1		0	0	
6	-1		0	0	
7	-1		0	0	

PE_7	state	f	E	C	Superstates
0	-1		0	0	
1	-1		0	0	
2	-1		0	0	
3	-1		0	0	
4	-1		0	0	
5	-1		0	0	
6	-1		0	0	
7	-1		0	0	

	extraa	CSuperstates	Extra
1	0.45	{(1,2,1), (1,2,2), (1,2,3), (2,2,1), (2,2,2), (2,2,3)}	{(1,1,1), (1,1,2), (1,1,3), (2,1,1), (2,1,2), (2,1,3)}
2	0.39	{(2,1,1), (2,2,1), (1,2,2), (2,2,2), (1,2,3), (2,2,3)}	{(1,1,1), (1,1,2), (1,1,3), (1,2,1), (2,1,2), (2,1,3)}
3	0.31	{(1,1,2), (1,2,2), (2,1,2), (2,2,2), (1,1,3), (1,2,3), (2,1,3), (2,2,3)}	{(1,1,1), (1,2,1), (2,1,1), (2,2,1)}

PartitionCount	3

D.6 Iteration of the GenerateSuperstates Routine

The GenerateSuperstates routine is a recursive one that generates all of the superstates of a given substate and returns that information, via a reference parameter, in a set. Iteration of the GenerateSuperstates Routine is illustrated by generating all of the superstates of the state ordering (0,0,2).

routine called	word	position	Superset
GenerateSuperstates	(0,0,2)		{

routine called	word	position	Superset
generate	(0,0,2)	1	
generate	(1,0,2)	2	(1,1,2), (1,2,2)
generate	(1,2,2)	3	
generate	(1,0,2)	3	
generate	(2,0,2)	2	(2,1,2), (2,2,2)
generate	(2,3,2)	3	
generate	(2,0,2)	3	
generate	(3,0,2)	2	
generate	(3,1,2)	3	
generate	(3,2,2)	3	
generate	(3,3,2)	3	}

D.7 Iteration of the Parallelization of the Reconstruction Process

The parallel algorithm for the reconstruction process is iterated using the partition obtained in section D.6. The results of this iteration follows.

iteration	init	1	2	3	4	5	6	7
$\hat{f}(000)$	0.083	0.065	0.066	0.067	0.069	0.071	0.072	0.074
$\hat{f}(001)$	0.083	0.075	0.081	0.084	0.085	0.086	0.087	0.087
$\hat{f}(002)$	0.083	0.085	0.086	0.085	0.085	0.084	0.084	0.083
$\hat{f}(010)$	0.083	0.052	0.042	0.038	0.036	0.035	0.034	0.034
$\hat{f}(011)$	0.083	0.065	0.059	0.056	0.055	0.054	0.053	0.053
$\hat{f}(012)$	0.083	0.095	0.094	0.095	0.095	0.096	0.096	0.097
$\hat{f}(100)$	0.083	0.073	0.079	0.083	0.085	0.086	0.088	0.088
$\hat{f}(101)$	0.083	0.071	0.072	0.073	0.073	0.074	0.074	0.073
$\hat{f}(102)$	0.083	0.051	0.046	0.044	0.042	0.041	0.040	0.039
$\hat{f}(110)$	0.083	0.120	0.123	0.122	0.120	0.118	0.116	0.115
$\hat{f}(111)$	0.083	0.129	0.128	0.127	0.127	0.126	0.126	0.127
$\hat{f}(112)$	0.083	0.119	0.124	0.126	0.128	0.129	0.130	0.131

iteration	8	9	10	11	12	13	14	15
$\hat{f}(000)$	0.075	0.075	0.076	0.077	0.077	0.077	0.078	0.078
$\hat{f}(001)$	0.087	0.088	0.088	0.088	0.088	0.088	0.088	0.088
$\hat{f}(002)$	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
$\hat{f}(010)$	0.033	0.033	0.032	0.032	0.032	0.032	0.032	0.032
$\hat{f}(011)$	0.053	0.052	0.052	0.052	0.052	0.052	0.052	0.052
$\hat{f}(012)$	0.097	0.097	0.097	0.097	0.097	0.097	0.097	0.097
$\hat{f}(100)$	0.089	0.089	0.090	0.090	0.090	0.090	0.090	0.091
$\hat{f}(101)$	0.073	0.073	0.073	0.073	0.072	0.072	0.072	0.072
$\hat{f}(102)$	0.039	0.039	0.038	0.038	0.038	0.038	0.038	0.038
$\hat{f}(110)$	0.114	0.113	0.112	0.111	0.11	0.111	0.110	0.110
$\hat{f}(111)$	0.127	0.127	0.127	0.127	0.128	0.128	0.128	0.128
$\hat{f}(112)$	0.131	0.131	0.132	0.132	0.132	0.132	0.132	0.132

iteration	16	17	18	19
$\hat{f}(000)$	0.078	0.078	0.078	0.079
$\hat{f}(001)$	0.088	0.088	0.088	0.088
$\hat{f}(002)$	0.083	0.083	0.083	0.083
$\hat{f}(010)$	0.031	0.031	0.031	0.031
$\hat{f}(011)$	0.052	0.052	0.052	0.052
$\hat{f}(012)$	0.097	0.097	0.097	0.097
$\hat{f}(100)$	0.091	0.091	0.091	0.091
$\hat{f}(101)$	0.072	0.072	0.072	0.072
$\hat{f}(102)$	0.037	0.037	0.037	0.037
$\hat{f}(110)$	0.110	0.110	0.110	0.109
$\hat{f}(111)$	0.128	0.128	0.128	0.128
$\hat{f}(112)$	0.133	0.133	0.133	0.133

D.8 Iteration of the Parallelization of a Greedy Algorithm for a Generalization of the Unbiased Reconstruction

The parallelization of a greedy algorithm for a generalization of the unbiased reconstruction is iterated using the example structure system. The results of the iteration follows. Note that the results at each iteration is the same as the iteration of the sequential algorithm. This is due to the use of the same maximizing function to choose which β to add to the set D.

Iteration	1	2	3	4	5	6	7	8	9
State added	$^{12}(11)$	$^{23}(10)$	$^{13}(12)$	$^{23}(11)$	$^{13}(11)$	$^{23}(12)$	$^{12}(10)$	$^{13}(01)$	$^{13}(02)$
$\hat{f}(000)$	0.07	0.07	0.08	0.09	0.09	0.09	0.08	0.08	0.08
$\hat{f}(001)$	0.07	0.07	0.08	0.09	0.09	0.09	0.08	0.09	0.09
$\hat{f}(002)$	0.07	0.07	0.08	0.09	0.09	0.09	0.08	0.08	0.08
$\hat{f}(010)$	0.07	0.05	0.05	0.03	0.03	0.03	0.03	0.03	0.03
$\hat{f}(011)$	0.07	0.07	0.08	0.04	0.05	0.05	0.05	0.05	0.05
$\hat{f}(012)$	0.07	0.07	0.08	0.09	0.09	0.1	0.1	0.1	0.1
$\hat{f}(100)$	0.07	0.07	0.08	0.09	0.09	0.09	0.09	0.09	0.09
$\hat{f}(101)$	0.07	0.07	0.08	0.09	0.07	0.07	0.07	0.07	0.07
$\hat{f}(102)$	0.07	0.07	0.06	0.04	0.04	0.04	0.04	0.04	0.04
$\hat{f}(110)$	0.12	0.09	0.1	0.11	0.11	0.11	0.11	0.11	0.11
$\hat{f}(111)$	0.12	0.14	0.16	0.14	0.13	0.13	0.13	0.13	0.13
$\hat{f}(112)$	0.12	0.14	0.12	0.13	0.13	0.13	0.13	0.13	0.13

Vita

Patti E. Iles received her Bachelors degree in Computer Science with a minor in Mathematics at Louisiana State University in May, 1989. During her undergraduate education, Ms. Iles represented Louisiana State University at the 1988 ACM South Central Regional Scholastic Programming contest which was held at Baylor University. She was an active member in the student chapters of the Association for Computing Machinery and the Upsilon Pi Epsilon honor society for computer scientists. Ms. Iles was an Arthur R. Choppin Memorial Honor recipient for outstanding scientific achievement in both 1988 and 1989. She was initiated into the honor societies of Phi Beta Kappa, Phi Kappa Phi, Upsilon Pi Epsilon, Mu Sigma Rho, and Kappa Delta Epsilon.

Patti E. Iles anticipates receiving her doctorate degree in Computer Science with a doctoral minor in Mathematics in May of 1995. During her graduate tenure at Louisiana State University, she has worked as a graduate assistant in the Department of Computer Science at Louisiana State University. Her assistantship duties have included designing, implementing, and maintaining a program for managing the department's fiscal budget, grading a wide range of classes, and teaching several classes for computer science majors and non-majors affording her the distinguished Louisiana State University Alumni Association Teaching Award for 1991. Her teaching repertoire includes Advanced Data Structures and Algorithm Analysis. In addition to her assistantship duties, Ms. Iles has instructed two correspondence courses for the Department of Independent Study at Louisiana State University.

Ms. Iles was involved in a variety of professional activities including judging the Louisiana Science and Engineering Fair in 1993 and 1994, judging the Louisiana State University Computer Science Department's High School Programming Contest in 1994 and 1995, representing the Louisiana State University at the Symposium for Female Students in Computing sponsored by the Computing Research Association and the National Science Foundation held in Washington, D.C. in 1993, participated as a panel member of experienced GTAs on "What New GTAs Need to Know" at an LSU Workshop Series for faculty and GTAs, presided over the LSU Computer Science Graduate Student Organization as President in 1991, represented Louisiana State University at the 1990 ACM South Central Region Scholastic Programming Contest, and attended a variety of seminars and conferences. Ms. Iles is a member of the Association for Computing Machinery, the ACM Special Interest Group for Computer Science Education, and the Special Interest Group for Algorithms, Control, and Theory. She also co-authored the Instructor's Manual for Introduction to Data Structures and Algorithms Analysis by Naps/Pothering which was published by West publishing in 1992.

Ms. Iles' educational focus has been in algorithm design and analysis, high performance computing, and software engineering.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Patti E. Iles

Major Field: Computer Science

Title of Dissertation: Parallelization of Reconstructability Analysis Algorithms

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

4/6/95